1-1-2013

Analyzing Massive Graphs in the Semi-streaming Model

Kook Jin Ahn
University of Pennsylvania, kookjin@seas.upenn.edu

Analyzing Massive Graphs in the Semi-streaming Model

Abstract
Massive graphs arise in a many scenarios, for example,

traffic data analysis in large networks, large scale scientific

experiments, and clustering of large data sets.

The semi-streaming model was proposed for processing massive graphs. In the semi-streaming model, we have a random

accessible memory which is near-linear in the number of vertices.

The input graph (or equivalently, edges in the graph)

is presented as a sequential list of edges (insertion-only model)

or edge insertions and deletions (dynamic model). The list

is read-only but we may make multiple passes over the list.

There has been a few results in the insertion-only model

such as computing distance spanners and approximating

the maximum matching.

In this thesis, we present some algorithms and techniques

for (i) solving more complex problems in the semi-streaming model,

(for example, problems in the dynamic model) and (ii) having

better solutions for the problems which have been studied

(for example, the maximum matching problem). In course of both

of these, we develop new techniques with broad applications and

explore the rich trade-offs between the complexity of models

(insertion-only streams vs. dynamic streams), the number

of passes, space, accuracy, and running time.

1. We initiate the study of dynamic graph streams.

We start with basic problems such as the connectivity

problem and computing the minimum spanning tree.

These problems are

trivial in the insertion-only model. However, they require
non-trivial (and multiple passes for computing the exact minimum
spanning tree) algorithms in the
dynamic model.
2. Second, we present a graph sparsification algorithm in the
semi-streaming model. A graph sparsification
is a sparse graph that approximately preserves
all the cut values of a graph.
Such a graph acts as an oracle for solving cut-related problems,
for example, the minimum cut problem and the multicut problem.
Our algorithm produce a graph sparsification with high probability
in one pass.
3. Third, we use the primal-dual algorithms
to develop the semi-streaming algorithms.
The primal-dual algorithms have been widely accepted
as a framework for solving linear programs
and semidefinite programs faster.
In contrast, we apply the method for reducing space and
number of passes in addition to reducing the running time.
We also present some examples that arise in applications
and show how to apply the techniques:
the multicut problem, the correlation clustering problem,
and the maximum matching problem. As a consequence,
we also develop near-linear time algorithms for the b-matching
problems which were not known before.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Sudipto Guha

Keywords
Algorithms, Graphs, Semi-streaming Model

Subject Categories
Computer Sciences

ANALYZING MASSIVE GRAPHS IN THE SEMI-STREAMING MODEL

Kook Jin Ahn

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2013

Supervisor of Dissertation

_____

Sudipto Guha, Associate Professor, Computer and Information Science

Graduate Group Chairperson

_____

Val Tannen, Professor, Computer and Information Science

Dissertation Committee

Rajeev Alur, Professor, Computer and Information Science

Sampath Kannan, Professor, Computer and Information Science

Aaron Roth, Assistant Professor, Computer and Information Science

Piotr Indyk, Professor, Computer Science and Artificial Intelligence Lab, MIT

ANALYZING MASSIVE GRAPHS IN THE SEMI-STREAMING MODEL

Acknowledgments

I would not have been able to finish this thesis without help from many people. It has been my great honor to have a chance to work with and learn from them. First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Sudipto Guha for being a wonderful advisor and providing me a great environment for research. Without his guidance and support, I would not have been able to achieve a lot of results in this thesis. I would like to thank Dr. Andrew McGregor for being my co-author, suggesting a great research topic, and a joyful atmosphere his jokes made.

I would also like to thank Dr. Rajeev Alur, Dr. Piotr Indyk, Dr. Sampath Kannan, and Dr. Aaron Roth for agreeing to be on my thesis committee. Their comments and questions allowed me improve this thesis. I would like to thank all the staffs in Penn CIS. Without their help on administrative work, I would not have been able to focus on research. I would like to thank fellow theory group students: Anand Bhalgat, Mickey Brautbar, Tanmoy Chakraborty, Zhiyi Hwang, and Sudeepa Roy.

This thesis is dedicated to my mother, Younghee Oh. She has been always supportive of me and my decision to study in US.

ABSTRACT

ANALYZING MASSIVE GRAPHS IN THE SEMI-STREAMING MODEL

Kook Jin Ahn

Sudipto Guha

Massive graphs arise in a many scenarios, for example, traffic data analysis in large networks, large scale scientific experiments, and clustering of large data sets. The semi-streaming model was proposed for processing massive graphs. In the semi-streaming model, we have a random accessible memory which is near-linear in the number of vertices. The input graph (or equivalently, edges in the graph) is presented as a sequential list of edges (insertion-only model) or edge insertions and deletions (dynamic model). The list is read-only but we may make multiple passes over the list. There has been a few results in the insertion-only model such as computing distance spanners and approximating the maximum matching.

In this thesis, we present some algorithms and techniques for (i) solving more complex problems in the semi-streaming model, (for example, problems in the dynamic model) and (ii) having better solutions for the problems which have been studied (for example, the maximum matching problem). In course of both of these, we develop new techniques with broad applications and explore the rich trade-offs between the complexity of models (insertion-only streams vs. dynamic streams), the number of passes, space, accuracy, and running time.

- We initiate the study of dynamic graph streams. We start with basic problems such as the connectivity problem and computing the minimum spanning tree. These problems are trivial in the insertion-only model. However, they require non-trivial (and multiple passes for computing the exact minimum spanning tree) algorithms in the dynamic model [7].

- Second, we present a graph sparsification algorithm in the semi-streaming model [4, 7]. A graph sparsification is a sparse graph that approximately preserves all the cut values of a graph. Such a graph acts as an oracle for solving cut-related problems, for example, the minimum cut problem and the multicut problem. Our algorithm produce a graph sparsification with high probability in one pass.

- Third, we use the primal-dual algorithms to develop the semi-streaming algorithms [5, 2, 6]. The primal-dual algorithms have been widely accepted as a framework for solving linear programs and semidefinite programs faster. In contrast, we apply the method for reducing space and number of passes in addition to reducing the running time.

We also present some examples that arise in applications and show how to apply the techniques: the multicut problem, the correlation clustering problem [6], and the maximum matching problem [5, 2]. As a consequence, we also develop near-linear time algorithms for the b-matching problems which were not known before.

Contents

Chapter 1

Introduction

1.1 Background

The streaming model has gained attention as a model for analyzing massive data sets. In the streaming model, we have sequential access to the input data and a random accessible memory of size sublinear (preferably logarithmic) in the input size. The computation proceeds as passes (preferably one) over the input data. There has been significant progress in some areas in the streaming model: for example, analyzing distributions (see [80, 62] and the references therein) and sampling from data streams (see [45, 64] and the references therein). However, similar progress has not been made for graph problems. Massive graphs arise in a many scenarios, such as traffic data analysis in large networks, large scale scientific experiments, and clustering of large data sets, to name a few.

Graph problems were considered in the streaming model and it was proven that even simple problems like connectivity problem requires $\Omega(n)$ space [58] (throughout this thesis n will denote the number of vertices and m will denote the number of edges). The lower bound on space motivates the introduction of the semi-streaming model [80] where the algorithm uses near-linear space in n, i.e., we allow the space larger than the lower bound by (1) a polylogarithmic factor in the input size, (2) a constant factor that depends on the approximation parameter $0 < \varepsilon \ll 1$, and/or (3) a small polynomial factor, for example, $n^{0.1}$.

In the following sections, we formally define the semi-streaming model and review the previous research results in the semi-streaming model.

1.1.1 The Semi-streaming Model

In the streaming model, the input data is a sequence of items. We consider data streams for graph problems or *graph data streams*. We consider two types of graph data streams: *insertion-only graph stream* and *dynamic graph stream*.

Definition 1.1.1 (Insertion-only Graph Stream)**.** In this model, a stream $\langle a_1, \ldots, a_t \rangle$ where $a_k \in [n] \times [n]$ defines a multi-graph $G = (V, E)$ where $V = [n]$ and the multiplicity of an edge (i, j) equals the number of occurrences of (i, j). $\square$

Definition 1.1.2 (Dynamic Graph Stream)**.** In this model, a stream $S = \langle a_1, \ldots, a_t \rangle$ where $a_k \in [n] \times [n] \times \{-1, 1\}$ defines a multi-graph $G = (V, E)$ where $V = [n]$ and

the multiplicity of an edge (i, j) equals

$$A(i, j) = |\{k : a_k = (i, j, +)\}| - |\{k : a_k = (i, j, -)\}| \ .$$

We assume that the edge multiplicity is non-negative and that the graph has no self-loops. $\square$

In the insertion-only model, we can maintain some information on the input graph before the end of the stream and based on the information, adapt the algorithm for the rest of the stream. For example, suppose that we want to test the connectivity of a graph. If we already know that two vertices u and v are connected, dropping (u, v) from the stream does not affect the output and therefore, the algorithm can safely ignore (u, v) in the rest of the stream. In the dynamic model, such an approach does not work because it is possible to erase all the inserted edges. We also define weighted graph streams as follows:

Definition 1.1.3 (Insertion-only Weighted Graph Stream)**.** In this model, a stream $\langle a_1, \ldots, a_t \rangle$ where $a_k \in [n] \times [n] \times [W]$ defines a multi-graph $G = (V, E)$ where $V = [n]$, W is the maximum edge weight, and the multiplicity of an edge (i, j, w) equals the number of occurrences of (i, j, w). $\square$

Definition 1.1.4 (Dynamic Weighted Graph Stream)**.** In this model, a stream $S = \langle a_1, \ldots, a_t \rangle$ where $a_k \in [n] \times [n] \times [W] \times \{-1, 1\}$ defines a multi-graph $G = (V, E)$ where $V = [n]$, W is the maximum edge weight and the multiplicity of an edge (i, j, w) equals

$$A(i, j, w) = |\{k : a_k = (i, j, w, +)\}| - |\{k : a_k = (i, j, w, -)\}| \ .$$

We assume that the edge multiplicity is non-negative and that the graph has no self-loops. □

In this thesis, we assume that edge weights are polynomially bounded in the number of vertices. Also, note that the definition of dynamic weighted graph stream does not support changing edge weights. For example, if we insert an edge $(i, j, 7)$, we cannot delete $(i, j, 3)$ to make its weight 4. Given a graph stream, we define the semi-streaming model as follows:

Definition 1.1.5 (The Semi-streaming Model)**.** The *semi-streaming model* is the streaming model for graph streams where algorithms use space that is near-linear in n but sublinear in m.

There are several important factors that define the quality of an algorithm in the semi-streaming model:

1. **Space:** The most central constraint of all streaming algorithms is the space requirement. The preferable result in this context is $O(n \operatorname{poly}(\frac{1}{\varepsilon}, \log n))$ but some algorithms may have exponential dependency on $\frac{1}{\varepsilon}$ or n^{1+c} space for some $0 < c < 1$.

2. **Passes:** The number of passes required by an algorithm is arguably the most highlighted aspect of an algorithm in the semi-streaming model. The preferable result is a single-pass algorithm since it expands the application domain of the algorithm. For example, single-pass algorithms may function as building blocks

of another algorithm without increasing the overall number of passes. So this is important in the context of dynamic streams.

3. **Accuracy:** Since the space is restricted in the semi-streaming model, we cannot hope for exact optimal solution for most problems. So the approximation factor (or accuracy) is also an important quality of an algorithm. The most preferable result is $(1-\varepsilon)$-approximation $(0 < \varepsilon \leq 1)$.

4. **Running Time:** Another factor we consider for analyzing semi-streaming algorithms is its running time. The semi-streaming model is motivated by massive graphs and algorithms are less useful if they do not run in near-linear time (throughout this thesis, near-linear time means $O(C_\varepsilon m \operatorname{polylog} n)$ time).

We also investigate the trade-offs between number of passes, accuracy, space, and running time. The most interesting trade-offs arises between space and number of passes, for example, there are algorithms that run in $O(p)$ passes and use $n^{1+1/p}$ space. These translate to $O(\log n)$-pass algorithms in the semi-streaming model as well as constant-pass algorithms when we allow space (polynomially) slightly larger than lower bound (for example, $n^{1.1}$).

1.1.2 Overview of Previous Research

Early results in graph streams mainly address statistical aspects of graphs, e.g., counting triangles [23, 15, 63] and analyzing degree sequences [29, 24] The semi-streaming

model became formalized after $\Omega(n)$ space lower bound was proven for problems, for example, determining if a graph is connected [58].

Results in the insertion-only semi-streaming model prior to this thesis mainly consisted of distance estimation [43, 44, 16, 40] and the maximum matching problem [43, 74, 95, 42, 39, 38].

Most known distance estimation algorithms in the semi-streaming model produce *spanners*. Once constructed, a spanner works as an oracle for distance-related problems such as shortest path problem. Many single-pass algorithms have been studied [43, 44] as well as multi-pass algorithms [16, 40].

Definition 1.1.6 (Distance Spanner)**.** A subgraph $H = (V, E')$ of $G = (V, E)$ is a (α, β)-spanner (of G) if, for any pair of vertices $u, v \in V$, $d_G(u, v) \leq d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$. where $d_G(u, v)$ denotes the distance between u and v in G. □

The *maximum matching problem* is the most extensively studied problem in the semi-streaming model and shows the rich trade-offs between number of passes, approximation factor, and space. For the maximum cardinality matching (MCM), Feigenbaum et al. [43] presented $(2/3-\varepsilon)$-approximation algorithm which was later improved to $(1-\varepsilon)$-approximation by McGregor [74]. This algorithm repeatedly finds augmenting paths through random mapping of vertices. However, the number of passes used by this result is exponential in $1/\varepsilon$. Eggert et al. [39] improved the number of passes to polynomial in $1/\varepsilon$, but this only applies to bipartite graphs. On the other hand, finding even a single augmenting path for the maximum weighted matching (MWM)

is difficult in the semi-streaming model and single-pass constant-factor approximation algorithms have been studied [43, 95, 42].

The lower bound results in the semi-streaming model mainly address $\Omega(n)$ space for a variety of problems [58]. Buchsbaum et al. [22] have proven a number of lower bounds for determining if there exists any pair (u, v) of vertices such that the number of vertices that are neighbors of both u and v exceeds a given threshold. The lower bound varies from $\Omega(n)$ to $\Omega(n^2)$ depending on various aspects of the problem and input. For example, two-sided error algorithm requires $\Omega(n^{1.5})$ space if the threshold is $o(n)$ while it requires $\Omega(n^2)$ space if the threshold is $\Theta(n)$. Lower bounds for distance spanners are also interesting and give space-accuracy trade-offs. Feigenbaum et al. [44] showed $\Omega(n^{1+1/\gamma})$ lower bound to approximate the distance between two vertices upto factor $1/\gamma$. Woodruff [94] showed that any $(1, 2k-1)$-spanner has $\Omega(\frac{1}{k}n^{1+1/k})$ edges.

1.2 Contributions

1.2.1 General Directions of Research

In this thesis, we investigate three directions of research for solving more complex problems in the semi-streaming model and explore the rich trade-offs between the complexity of models, the number of passes, space, accuracy, and running time. The directions can be categorized as **models**, **tools**, and **techniques**.

(i) Models: We initiate the study of dynamic graph streams. In the insertion-only streaming model, the deletions of input items cannot be processed without a huge number of passes. So the study of dynamic graph streams is essential for applications which require the updates on the input graph. We start with basic problems such as the connectivity problem and computing the minimum spanning tree (MST). These problems are trivial in the insertion-only model. However, they require non-trivial algorithms (and multiple passes for computing the exact MST) in the dynamic model [7]. This demonstrates the richness of the problem space in the semi-streaming model.

(ii) Tools: A roadblock in solving a wide range of problems in the semi-streaming model is lack of tools that are useful in many graph problems. We investigate distance spanners and graph sparsification. Recall spanners are defined in Definition 1.1.6. We can use a distance spanner as a building block in many distance-related problems.

A graph sparsification is a sparse graph that approximately preserves all the cut values of a graph. Formally, it is defined as follows:

Definition 1.2.1 (Graph Sparsification)**.** A graph $H = (V, E', w')$ is a $(1 \pm \varepsilon)$-sparsification of $G = (V, E, w)$ if, for any cut $(A, V \setminus A)$, $(1 - \varepsilon)c_G(A, V \setminus A) \leq c_H(A, V \setminus A) \leq (1 + \varepsilon)c_G(A, V \setminus A)$ where $c_G(A, V \setminus A)$ is the cut value of $(A, V \setminus A)$ in G. $\square$

A graph sparsification has a similar flavor as a distance spanner except it approximates

cut values instead of distance. It acts as an oracle for solving cut-related problems, for example, the minimum cut problem and the multicut problem. We present a one-pass graph sparsification algorithm in the semi-streaming model that succeeds with high probability [4, 7, 8]. The algorithm is built upon the k-connectivity algorithm and works in the dynamic model using a single pass. Naturally, the algorithm performs better in the insertion-only model.

(iii) Techniques: The tools we previously mentioned – distance spanners and graph sparsification – are tied to specific properties – distances and cut values. However, in order to solve complex graph problems, we need general techniques (or approaches) to design algorithms.

We investigate adopting convex optimization techniques such as linear program (LP) and semidefinite program (SDP) to the semi-streaming model. We use primal-dual algorithms to develop semi-streaming algorithms [5, 2, 6]. Primal-dual algorithms like the multiplicative weights update method [11] has been widely accepted as a framework for solving LPs and SDPs faster. In contrast, we apply the method for reducing space and number of passes in addition to reducing the running time.

1.2.2 Summary of Results

In this section, we discuss the general directions in further details and summarize corresponding results in this thesis.

Linear Measurements and Dynamic Graph Streams: For dynamic graph streams, we solve problems through linear measurements.

Definition 1.2.2 (Linear Measurements)**.** A vector $\mathcal{S}X$ where $X \in \mathbb{R}^n$ is an input vector and $\mathcal{S} \in \mathbb{R}^{d \times n}$ is a (random) linear projection. □

The idea is to take linear measurements while reading the stream and solve the problem offline from the linear measurements. Because of linearity of the measurements, the update of measurements (for an insertion or a deletion) can be performed without looking at other items in the data stream. Therefore, the algorithms based on linear measurements are suitable for dynamic graph streams. In addition, such algorithms can be used in many distributed models since the computation of linear measurements is embarrassingly parallelizable.

Our algorithms build upon sampling algorithms which are based on linear measurements. Specifically, we use ℓ_0-sampling which returns one non-zero coordinate given a vector when the vector is specified as a stream of updates. We use multiple ℓ_0-sampling in parallel by concatenating linear projections for ℓ_0-sampling. We also use random hash functions and pseudo random generators. Such tools are useful when we create a (virtual) stream by subsampling a large number of edges (for example, subsampling edges with probability 1/2). Note that having a single-pass algorithm based on linear measurements makes it easy to prove such an algorithm using the argument in [61].

We present single-pass semi-streaming algorithms for testing connectivity (span-

ning forest), k-connectivity, and bipartiteness. We also present $O(p)$-pass $n^{1+1/p}$-space algorithms for minimum spanning tree and maximal matching. Finally, we present algorithms for constructing distance spanner in the dynamic model. However, our algorithm takes multiple passes over the data.

Graph Sparsification: In this thesis, we present a single-pass semi-streaming graph sparsification algorithm which builds upon the k-connectivity algorithm. The algorithm works for the dynamic model and has better performance in the insertion-only model. As in the case of distance spanners, the graph sparsification has many applications. For example, the multi-cut problem and the correlation clustering problem can be made space efficient with the use of the graph sparsification. We construct a sparsification for the input graph and solve the problem for the sparsified graph instead of the input graph. The approach gives a single-pass semi-streaming algorithm. Our maximum matching algorithm (for non-bipartite graphs) also uses the graph sparsification as its subroutine.

Primal-Dual Algorithms: We investigate the use of primal-dual based algorithms such as the multiplicative weights update method [11] and the fractional packing/covering framework [84] for solving a subclass of LPs and SDPs on graphs. These primal-dual algorithms use an oracle to progressively improve the feasibility of the primal linear program, but uses a (guessed) value of the optimal solution. If the oracle does not fail to provide these improvements within a predetermined number

of iterations, we are guaranteed an approximately feasible primal solution. If the oracle fails to provide an improvement, the oracle proves that the guessed value of the optimal solution is incorrect by either (i) give us a feasible dual solution or (ii) prove that the oracle would have produced an improvement if the guessed value was correct. Both these properties together gives an overall scheme to find approximately feasible solutions of LPs and SDPs.

Designing an efficient separation oracle is not always trivial even without any constraint on space. However, even if we could design an efficient oracle, the overall scheme to obtain a good semi-streaming algorithm faces a number of roadblocks. First, primal-dual algorithms typically require super-constant number of iterations (to determine feasibility) — this translates to a super-constant number of passes. Reducing the number of passes to a constant often requires significant effort and new ideas. It is non-trivial to simultaneously ensure that enough global progress is being made per pass, yet the computation in a pass is local (and in small space).

We present multiple applications of the primal-dual algorithms: (1) the maximum matching problem [5, 2], (2) the multicut problem, and (3) the correlation clustering [6],

The maximum matching problem demonstrates the difficulties in such an approach. The maximum matching problem is well-studied in the random access memory (RAM) model and there is a near-linear time (in the number of vertices) approximation scheme for the maximum weighted matching problem [34]. In the semi-

streaming model, the problem has received a lot of attention but a similar result has not been shown. For MCM, previous results require a huge number of passes [74] or the input graph must be bipartite [39, 38]. This is because they use the approach of augmenting paths. For MWM, the approximation factors are not optimal [43, 74, 95, 42]. We use the linear programming approach to improve and unify the previous results. The maximum matching problem requires deep understanding of the problem structure (such as laminarity of the optimal solution) and introduces complications in combining building blocks for an overall algorithm.

On the other hand, the multicut problem and the correlation clustering problem demonstrate the use of graph sparsification and linear programming in the semi-streaming model. We translate existing algorithms [52, 33, 91] to the semi-streaming model with (relatively) straight-forward modifications.

1.3 Related Work

There are a variety of notions of approximating graphs other than distance spanner and sparsification. Spielman and Teng [89] generalizes the notion of the (cut) sparsification to the spectral sparsification which preserves $\sum_{i,j} w_{ij}(x_i - x_j)^2$ for any vector $\mathbf{x} \in [0,1]^n$.[1] The result has been improved later in [88, 18] and also a semi-streaming algorithm has been presented in [68]. Another notion of graph approximation is vertex sparsification which is introduced by Moitra [77]. The vertex sparsification

[1]The cut sparsification preserves $\sum_{i,j} w_{ij}(x_i - x_j)^2$ for any vector $\mathbf{x} \in \{0,1\}^n$.

preserves minimum cuts separating all partitions of terminals (a subset of vertices) rather than all cuts in the graph. The result has been improved later in [71, 26]. The vertex sparsification is smaller than the cut sparsification but has a larger approximation factor [71]. In addition, the concept of spanners is not limited to distance. For example, transitive-closure spanners preserves a set of transitive closures [21]. Transitive-closure spanners have been implicitly studied in a number of areas including access control, property testing, and data structures. See [21] for the more detailed history of spanners.

The matching problem has a rich literature, see [37, 50, 59, 76, 35, 36], as well as fast, near-linear time approximation algorithms [65, 85, 93, 83, 34]. These results cover both MCM and MWM in bipartite graphs as well as general graphs.

The current best algorithms for the b–Matching all require quadratic in running time. The following is the list of the current best algorithms;

- Gabow [49] gave an $O(nm \log n)$ algorithm for the unweighted ($w_{ij} = 1$, mentioned as cardinality problem in [49]) capacitated problem.
- For the weighted uncapacitated case Anstee [10] gave an $O(n^2 m)$ algorithm; an $\tilde{O}(m^2)$ algorithm is in [49].
- Lechtford et al. [73], building on Padberg and Rao [82], gave an $O(n^2 m \log(n^2/m))$ time algorithm for the weighted, uncapacitated/capacitated problem.
- In terms of approximation algorithms there has been a sequence of results that have culminated in a $O(m \max_i b_i)$ time $\frac{1}{2}$ approximation for weighted unit

capacity b–Matching [75]. Distributed algorithms with weaker approximation guarantees are discussed in [69].

It is not too hard to see that if we were to copy each vertex b_i times then the b–Matching problems reduce to maximum weighted matching. However the size of the graph increases drastically[2]. A compressed representation was introduced in [49] to avoid this blowup. But this representation is not approximation preserving – see the discussion in [60]. On the other hand, if we do not copy the vertices then standard augmentation path based techniques (including recent elegant results for maximum matching such as [34, 35, 36]) do not immediately apply. The augmentation structure needed for b–Matching are not just blossoms (which are standard in matching literature) but also blossoms with "petals/arms" or forests that are attached to the blossom, see the discussion in [79]. We note that b–Matching is a fundamental problem with a long and rich history in combinatorial optimization, see [86, Chapters 31–33] for a brief history.

1.4 Organization

This thesis consists of three parts.

1. In the first part, we discuss basic problems including testing connectivity(finding spanning forest), finding minimum spanning tree, and the maximal matching

[2]Consider a star graph where the central node has $b_i = n$ and the leaf nodes have $b_i = 1$. The replication of the central node will make the number of edges n^2.

problem in Chapter 3 and distance spanners in Chapter 4. The algorithms for the dynamic model appeared in [7, 8]. The graph sparsification and maximum matching results build upon the connectivity and maximal matching results.

2. In the second part, we discuss the graph sparsification. The graph sparsification in the semi-streaming model appeared in [4]. The algorithm in [4] was for the insertion-only model. We extended the result to the dynamic stream model in [8]. The algorithm in [8] is simpler and applicable to both models if a spanning forest algorithm is given as a black box. Hence, we only explain the algorithm in [8].

3. The third part focuses on convex optimization, such as LPs and SDPs in the context of the semi-streaming model. We consider a variety of applications such as maximum matching, b-matching, multi-cut, and correlation clustering. These algorithms build upon graph sparsification and maximal matching algorithms. In Chapter 6, we discuss the general approaches to LPs and SDPs with algorithms for bipartite maximum matching. These results first appeared in [5]. In Chapters 7, we extend the results to b–Matching in non-bipartite graphs which appeared in [2]. In Chapters 9, we again discuss b–Matching in non-bipartite graphs but using a dual-primal approach to LPs. The results first appeared in [3]. In Chapter 10, we discuss the multicut and correlation clustering problem.

Chapter 2

Preliminaries

2.1 ℓ_p-Sampling

ℓ_p-sampling is a problem that has recently enjoyed considerable attention [45, 30, 78, 64, 51]. Consider a stream $S = \langle s_1, \ldots, s_t \rangle$ where each $s_j = (u_j, \Delta_j) \in [n] \times \mathbb{R}$ and the aggregate vector $\mathbf{x} \in \mathbb{R}^n$ defined by this stream of updates, i.e., $x_i = \sum_{j:u_j=i} \Delta_j$.

Definition 2.1.1 (ℓ_p-Sampling)**.** An (ϵ, δ) ℓ_p-sampler for $\mathbf{x} \neq 0$ fails with probability at most δ and otherwise returns some $i \in [n]$ with probability in the range $\left[\frac{(1-\epsilon)|x_i|^p}{\ell_p^p(\mathbf{x})}, \frac{(1+\epsilon)|x_i|^p}{\ell_p^p(\mathbf{x})}\right]$, where $\ell_p^p(\mathbf{x}) = \sum_{i\in[n]} |x_i|^p$.

We use ℓ_p-sampling technique to sample an edge in the input graph. Solving problems in the dynamic graph streams often requires a technique to output an edge in the input graph without remembering all the edges in the graph. For example, finding a spanning forest requires such a technique since its output consists of edges

in the input graph. Suppose that we interpret the input graph as a vector in which each coordinate corresponds to a pair of vertices and its value indicates whether there exists an edge between the pair. By using ℓ_p-sampling for this vector, we can sample an edge from the input graph.

The next lemma, due to Jowhari et al. [64], summarizes the state-of-the-art result for $p \in \{0, 1\}$.

Lemma 2.1.2. *There exists linear sketch-based algorithms that perform ℓ_p-sampling using:*

1. $O(\log^2 n \log \delta^{-1})$ *space for* $p = 0$. *Note we may set* $\epsilon = 0$ *in this case.*

2. $O(\epsilon^{-1} \log \epsilon^{-1} \log^2 n \log \delta^{-1})$ *space for* $p = 1$.

We describe the ℓ_0-sampling algorithm of [64] in Algorithm 2. The algorithm relies on the unique element data structure [45] which is described in Algorithm 1.[1] The unique data structure takes a sequence of updates on the input vector and returns a coordinate and its value only if there is exactly one non-zero coordinate. Otherwise, it returns $\perp$. The ℓ_0-sampling algorithm samples "coordinates" with probability $1/2^k$ for $k = 0, 1, \cdots, \lfloor \log n \rfloor$. With constant probability, the algorithm will sample exactly one non-zero coordinate from the input vector and the corresponding unique element data structure will report the non-zero coordinate. We can also achieve a smaller failure probability δ by repeating the algorithm $\log(1/\delta)$ times in parallel. There is

[1] In [64], the authors used an exact recovery of sparse vector. However, the unique element data structure is sufficient for our purpose.

one remaining issue. It takes $O(n)$ space to generate and store I_k if we pick coordinates fully independently. However, we can replace the random bits with Nisan's random bits generator [81] which will increase the memory requirement by factor of $O(\log n)$.

Algorithm 1 The Unique Element Data Structure [45]

Update(u_i, Δ_i)	Report
$c_0 \leftarrow c_0 + \Delta_i$.	**if** $c_0 \cdot c_2 - c_1^2 \neq 0$ or $c_0 = 0$ **then**
$c_1 \leftarrow c_1 + u_i \cdot \Delta_i$.	**return** $\perp$ **(fail)**.
$c_2 \leftarrow c_2 + u_i^2 \cdot \Delta_i$.	**else**
	return $(c_1/c_0, c_0)$.
	end if

Algorithm 2 ℓ_0-Sampling Algorithm [64]

1: Let I_k for $k = 0, 1, \cdots, \lfloor \log k \rfloor$ be a subset of coordinates $[n]$ such that each coordinate is independently chosen with probability $1/2^{-k}$. ($I_0 = [n]$).

2: For each k, construct the unique element data structure of the input vector restricted to the coordinates in I_k.

3: The algorithm fails if the unique element data structure returns $\perp$ for all k.

Note that the whole ℓ_0-sampling algorithm is in fact based on linear measurements and the measurement matrix is generated from a random generator of $O(\text{polylog}\, n)$-size description.

2.2 Graph Sparsification

Karger [67] initiated a study on constructing a graph that contains a smaller number of edges than a given graph while preserving cut values of the given graph. The following lemma summarizes the main result.

Lemma 2.2.1 (Uniform Sampling Lemma [67])**.** *Given an undirected unweighted graph G, let λ be the minimum cut value. If we sample each edge with $p = \Omega\left(\lambda^{-1}\epsilon^{-2}\log n\right)$ and assign weight $1/p$ to sampled edges, every cut value is preserved within $1 \pm \epsilon$ factor with high probability.*

Benczür and Karger [19] used Lemma 2.2.1 to construct a graph sparsification algorithm. Fung et al. [46] generalized the algorithm for the graph sparsification and presented an algorithm that uses the standard edge connectivity (the minimum cut value which the edge crosses) using the framework. The following is the result.

Definition 2.2.2 (Fung et al. [46])**.** For an edge $e = (u, v)$, let λ_e be the cut value of the minimum $u-v$ cut. A skeleton G_ϵ is a graph that is constructed by sampling each edge with probability $p_e \geq \min\left\{253\lambda_e^{-1}\epsilon^{-2}\log^2 n, 1\right\}$ independently and assigning weight $1/p_e$ for sampled edges. For a cut C, let c and $\hat{c}$ be the cut value of C in G and G_ϵ respectively. We write $G_\epsilon \in (1 \pm \epsilon)G$ if $\frac{1}{1+\epsilon}c \leq \hat{c} \leq (1+\epsilon)c$ for every cut C.

Theorem 2.2.3 (Fung et al. [46])**.** *A skeleton $G_\epsilon \in (1 \pm \epsilon)G$ with high probability.*

The framework in [46] applies to other definitions of connectivities not just edge connectivity. However, we are not going to use the other results.

2.3 Primal-Dual Frameworks

In this section, we summarize the techniques for solving convex optimization: "primal-dual" algorithms. Primal dual algorithms have become an increasingly important technique and are used in many disparate fields, with a multitude of variations and applications; see the excellent survey of Arora, Hazan and Kale [4]. In this thesis, "primal-dual" terminology to refer to algorithms that provide fast and approximate solutions to linear programs (LPs), fractional packing problems, and semidefinite programs (SDPs). We mainly use the multiplicative weights update method surveyed by Arora, Hazan, and Kale [11] and the framework by Plotkin, Shmoys, and Tardos [84] among many variations. Although they were originally developed for faster approximation, they are also useful for solving linear programs in small space. We also summarize a framework for the semidefinite programming (SDP) by Steurer [90] and Arora and Kale [12].

Algorithms that use these frameworks consist of two-party games between the framework and an oracle. The oracle is a problem-specific part of the algorithm. The framework maintains a primal candidate and computes a dual candidate. Although these candidates have the form of primal and dual variables, they are often infeasible and therefore, not solutions. The oracle's role is very similar to the separation oracle [56]. The oracle interprets the dual candidate as weights of constraints and finds a primal witness which satisfies constraints of primal LP (or SDP) on (weighted) average. The primal witness also has the form of primal variables like the primal

candidate and provides a direction in which the primal candidate can be "improved". The framework updates its primal candidate and dual candidate. If the primal witness violates a constraint, the framework increases the corresponding variable in the dual candidate in a multiplicative way. The intuition behind it is that if the oracle repeatedly returns primal witnesses which violate a constraint, dual variable corresponding to the constraint will increase exponentially and therefore, the oracle has no choice but to satisfy the constraint.

2.3.1 Multiplicative Weights Update Method

In this section, we briefly explain the multiplicative weights update method; we follow the discussion presented by Arora, Hazan, and Kale [11]. Suppose that we are given the following LP, its dual LP, and a guess of the optimal solution α, where $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$:

$$\text{LP:} \begin{cases} \max & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t} & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0} \end{cases} \qquad \text{Dual LP:} \begin{cases} \min & \mathbf{b}^{\mathrm{T}}\mathbf{y} \\ \text{s.t} & \mathbf{A}^{\mathrm{T}}\mathbf{y} \geq \mathbf{c}, \quad \mathbf{y} \geq \mathbf{0} \end{cases}$$

The algorithm proceeds along the weak separation framework [56]. Suppose that the optimal solution is α. The **violation** of constraint i is $\mathbf{A}_i\mathbf{x} - b_i$. The complementary slackness conditions mandate that for an optimal solution that $y_i(\mathbf{A}_i\mathbf{x} - b_i) = 0$. One way to express the complementary slackness conditions into a single condition is to interpret the dual variables (which are always maintained as positive) as probabilities, and ask: *Is there a vector* $\mathbf{x}$ *which satisfies* $\mathbf{c}^{\mathrm{T}}\mathbf{x} = \alpha$*, such that the expected*

violation is at most δ? The vector $\mathbf{x}$, which is the answer to the question, is termed as a primal witness.

If the answer to the question posed to an oracle is "yes", and the probabilities were chosen such that constraints which had larger violations had larger probability mass; then we have a direction in which the feasibility of the primal candidate can be improved. The improvement is measured by a potential function, which is akin to the notion of dual gap.

If the answer is "no" (referred to as the failure of the oracle) — then we know that there is no "good direction" to improve the solution. This serves as a certificate that the LP (with the additional constraint that the solution is at least α) is not feasible. For example, a feasible dual solution which is less than α can be one such certificate. However since we are asking questions to the oracle that have an approximation parameter, the certificate is also approximate at best.

But, note that, neither of the above gives us a solution to the primal. *However we can produce a primal solution if we achieve two things, in addition to designing an oracle.*

- First, if we are careful in choosing the probabilities (which is what the multiplicative update framework achieves), then we have a way to extract a primal candidate which approximately satisfies all the primal constraints. In fact the solution will be the average of the primal witnesses found, and this average will approximately satisfy the primal constraints. It is easy to see that the average

satisfies $\mathbf{c}^T\mathbf{x} = \alpha$. Now in many situations, and for the problems in this thesis, a simple scaling (multiplying each coordinate by of this average vector by a constant c) can ensure primal feasibility and we have a c approximate dual solution.

- Second, note that the approximation also depends on the appropriate guess of the optimum solution, α. Therefore we need a way of verifying the feasibility of the parameter α. In this thesis we will achieve this by (i) creating a dual feasible solution which is at most $(1 + O(\delta))\alpha$, or (ii) implicitly proving that there is no admissible solution using a known constant factor approximation algorithm.

Since this thesis is regarding the application of the multiplicative weights update method in the streaming setting and not about the framework itself, we refer the reader to the original article of [11] for further discussion of the intuition behind the framework. In what follows, we provide a brief review of the main definitions and notation (Definition 2.3.1) and the meta-algorithm (Algorithm 3), and an extension (Theorem 2.3.2) of the main ingredient of [11] which is used in this thesis.

Definition 2.3.1. The multiplicative weights update method (Algorithm 3) proceeds in iterations and in iteration t finds a primal witness $\mathbf{x}^t$. We define $\mathbf{M}(i, \mathbf{x}^t) = \mathbf{A}_i\mathbf{x}^t - b_i$ to be the **violation** for primal constraint i in iteration t. The **expected violation** $\mathbf{M}(\mathcal{D}^t, \mathbf{x}^t)$ is the expected value of $\mathbf{M}(i, \mathbf{x}^t)$ when choosing i with probability proportional to u_i^t, i.e., $\sum_i \frac{u_i^t}{\sum_j u_j^t}\mathbf{M}(i, \mathbf{x}^t)$. The primal witness $\mathbf{x}^t$ is defined to be **admissible**

Algorithm 3 The Multiplicative Weights Update Method [11]

1: $u_i^1 = 1$ for all $i \in [n]$.

2: **for** $t = 1$ **to** T **do**

3: Given $\mathbf{u}^t$, the oracle returns an admissible primal witness $\mathbf{x}^t$. Note that $\mathbf{x}^t$ is not required to be feasible.

4: Let $\mathbf{M}(i, \mathbf{x}^t) = \mathbf{A}_i \mathbf{x}^t - b_i$ (for all i).

5: For all i, set $u_i^{t+1} = \begin{cases} u_i^t (1+\epsilon)^{\mathbf{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathbf{M}(i,\mathbf{x}^t) \geq 0 \\ u_i^t (1-\epsilon)^{-\mathbf{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathbf{M}(i,\mathbf{x}^t) < 0 \end{cases}$

6: **end for**

7: Output $\tilde{\mathbf{x}} = \left(\min_i \frac{b_i}{b_i + 4\delta}\right) \frac{1}{T} \sum_t \mathbf{x}^t$. Note, for use in this thesis $b_i \geq 1$. This step is dependent on the specific problem, here given for matching.

if it satisfies

$$\mathbf{M}(\mathcal{D}^t, \mathbf{x}^t) \leq \delta, \quad \mathbf{c}^{\mathrm{T}} \mathbf{x}^t \geq \alpha, \quad \text{and} \quad \mathbf{M}(i, \mathbf{x}^t) \in [-\ell, \rho] \quad \forall i \in [n] = \{1, \ldots, n\}$$

for parameters of the oracle ℓ and ρ such that $0 < \ell \leq \rho$ The parameters ℓ, ρ will be constants for the oracles in this thesis; ρ is called the width parameter of the oracle. The parameters ϵ and T depend on ρ, ℓ, and δ. Note that admissibility does not imply feasibility and that the admissibility is a property of $(\mathbf{u}^t, \mathbf{x}^t)$.

Theorem 2.3.2. *Let $\delta > 0$ be an error parameter and $\epsilon = \min\{\frac{\delta}{4\ell}, \frac{1}{2}\}$. Let $\Upsilon_i^t = u_i^t / \sum_j u_j^t$ and let $\Psi_i \geq \Upsilon_i^t$ for all t. Suppose that the oracle returns admissible primal witnesses (See Definition 2.3.1) for $T = \frac{2\rho}{\delta\epsilon} \ln \max_i \frac{\Psi_i}{\Upsilon_i^1}$ iterations in the multiplicative weights update method, then for any constraint $1 \leq i \leq m$ we have:*

$(1-\epsilon)\sum_t \mathbf{M}(i,\mathbf{x}^t) \le \delta T + \sum_t \mathbf{M}(\mathcal{D}^t,\mathbf{x}^t)$. *Moreover* $\tilde{\mathbf{x}}$ *is a feasible solution of the LP.*

Proof. We analyze the algorithm using a potential function $\Phi^t = \sum_j u_j^t$. Let $\Upsilon_i^t = u_i^t/\Phi^t$. We assume we have an upper bound $\Psi_i \ge \Upsilon_i^t$. Note that Υ_i^t and Ψ_i are not used in [11]. In this theorem, we use $\Psi_i = 1$ for all i — which is obvious from the fact that all weights u_i^t are positive. We will use a smaller value of Ψ_i to strengthen the theorem later.

We rewrite the proof of [11] using Υ_i^t and Ψ_i. Observe that $(1-\epsilon)^{-x}$ and $(1+\epsilon)^x$ are convex (in x) for $0 < \epsilon \le \frac{1}{2}$. Therefore it follows that

$$(1-\epsilon)^{-x} \le (1+\epsilon x) \qquad \text{for } x \in [-1,0]$$

$$(1+\epsilon)^{x} \le (1+\epsilon x) \qquad \text{for } x \in [0,1]$$

since equality is achieved at the respective endpoints ($x = -1, 0$ for the first fact and $x = 0, 1$ for the second fact). From $M(i,\mathbf{x}^t)/\rho \in [-1,1]$ (notice $\ell \le \rho$) and the above facts we have:

$$\begin{aligned}
\Phi^{t+1} &= \sum_i u_i^{t+1} = \sum_{i:M(i,\mathbf{x}^t)<0} u_i^t(1-\epsilon)^{-M(i,\mathbf{x}^t)/\rho} + \sum_{i:M(i,\mathbf{x}^t)\ge 0} u_i^t(1+\epsilon)^{M(i,\mathbf{x}^t)/\rho} \\
&\le \sum_i u_i^t(1+\epsilon M(i,\mathbf{x}^t)/\rho) = \Phi^t + \frac{\epsilon}{\rho}\sum_i u_i^t M(i,\mathbf{x}^t) \\
&= \Phi^t + \frac{\epsilon\Phi^t}{\rho}\sum_i \frac{u_i^t}{\Phi^t} M(i,\mathbf{x}^t) \\
&= \Phi^t(1+\epsilon M(\mathcal{D}^t,\mathbf{x}^t)/\rho) \qquad \big(\text{Using the definition of } M(\mathcal{D}^t,\mathbf{x}^t)\big) \\
&\le \Phi^t e^{\left(\epsilon M(\mathcal{D}^t,\mathbf{x}^t)/\rho\right)}.
\end{aligned}$$

Therefore we can conclude that,

$$\Phi^{T+1} \leq \Phi^1 e^{\left(\epsilon \sum_{t=1}^{T} M(\mathcal{D}^t, \mathbf{x}^t)/\rho\right)}. \tag{2.3.1}$$

From the algorithm and the definitions of Ψ_i,

$$\begin{aligned} u_i^1 (1+\epsilon)^{\left(\sum_{t:M(i,\mathbf{x}^t)\geq 0} M(i,\mathbf{x}^t)/\rho\right)} \cdot (1-\epsilon)^{-\left(\sum_{t:M(i,\mathbf{x}^t)<0} M(i,\mathbf{x}^t)/\rho\right)} \\ = u_i^{T+1} \leq \Phi^{T+1}\Psi_i. \end{aligned} \tag{2.3.2}$$

From the definition of $\Phi^t, \Upsilon_i^t, \Psi_i$, we get $\Phi^1 = u_i^1/\Upsilon_i^1$. Using Φ^1 in equations (2.3.1) and (2.3.2), we get,

$$\begin{aligned} u_i^1 (1+\epsilon)^{\left(\sum_{t:M(i,\mathbf{x}^t)\geq 0} M(i,\mathbf{x}^t)/\rho\right)} \cdot (1-\epsilon)^{-\left(\sum_{t:M(i,\mathbf{x}^t)<0} M(i,\mathbf{x}^t)/\rho\right)} \\ \leq \frac{\Psi_i}{\Upsilon_i^1} u_i^1 e^{\left(\epsilon \sum_{t=1}^{T} M(\mathcal{D}^t, \mathbf{x}^t)/\rho\right)}. \end{aligned}$$

Applying the natural log function and simplifying we get:

$$\ln(1+\epsilon) \sum_{t:M(i,\mathbf{x}^t)\geq 0} M(i,\mathbf{x}^t) - \ln(1-\epsilon) \sum_{t:M(i,\mathbf{x}^t)<0} M(i,\mathbf{x}^t) \leq \rho \ln \frac{\Psi_i}{\Upsilon_i^1} + \epsilon \sum_{t=1}^{T} M(\mathcal{D}^t, \mathbf{x}^t).$$

Now $\ln(1+\epsilon) - \epsilon(1-\epsilon) \geq 0$; we have equality at $\epsilon = 0$ and the first derivative of the left hand side with respect to ϵ is positive for $\epsilon > 0$. Likewise (using the derivative, but only over the range $0 < \epsilon \leq \frac{1}{2}$) we have $\ln(1-\epsilon) + \epsilon(1+\epsilon) \geq 0$. Therefore using $\ln(1+\epsilon) \geq \epsilon(1-\epsilon)$ and $\ln(1-\epsilon) \geq -\epsilon(1+\epsilon)$ we get:

$$\begin{aligned} \frac{\rho}{\epsilon} \ln \frac{\Psi_i}{\Upsilon_i^1} + \sum_{t=1}^{T} M(\mathcal{D}^t, \mathbf{x}^t) &\geq (1-\epsilon) \sum_{t:M(i,\mathbf{x}^t)\geq 0} M(i,\mathbf{x}^t) + (1+\epsilon) \sum_{t:M(i,\mathbf{x}^t)<0} M(i,\mathbf{x}^t) \\ &= (1-\epsilon) \sum_{t=1}^{T} M(i,\mathbf{x}^t) + 2\epsilon \sum_{t:M(i,\mathbf{x}^t)<0} M(i,\mathbf{x}^t) \\ &\geq (1-\epsilon) \sum_{t=1}^{T} M(i,\mathbf{x}^t) - 2\epsilon\ell T \qquad \left(\text{From } M(i,\mathbf{x}^t) \geq -\ell\right) \end{aligned}$$

Selecting $T = \max_i \frac{2\rho}{\delta\epsilon} \ln \frac{\Psi_i}{\Upsilon_i^1}$, $\Psi_i = 1$, and $\epsilon = \min\left\{\frac{\delta}{4\ell}, \frac{1}{2}\right\}$, we obtain $(1-\epsilon)\sum_t \mathbf{M}(i, \mathbf{x}^t) \leq \delta T + \sum_t \mathbf{M}(\mathcal{D}^t, \mathbf{x}^t)$. Since $\frac{1}{T}\sum_t \mathbf{M}(i, \mathbf{x}^t) = \mathbf{M}(i, \frac{1}{T}\sum_t \mathbf{x}^t)$ and $\mathbf{M}(\mathcal{D}^t, \mathbf{x}^t) \leq \delta$ for all t, we have $\mathbf{M}(i, \frac{1}{T}\sum_t \mathbf{x}^t) \leq (1-\epsilon)^{-1}(2\delta) \leq 4\delta$ (dividing both the left and right side of the inequality by T) or $\mathbf{A}_i\left(\frac{1}{T}\sum_t \mathbf{x}^t\right) \leq b_i + 4\delta$. This translates to the fact that $\tilde{\mathbf{x}}$ satisfies $\mathbf{A}_i\tilde{\mathbf{x}} \leq b_i$. □

Setting $u_i^1 = 1$, $\Psi_i = 1$ and $\Upsilon_i^1 = \frac{1}{n}$ we get the following corollary.

Corollary 2.3.3 (A slight rewording of Corollary 3 in [11])**.** *Let $\delta > 0$ be an error parameter and $\epsilon = \min\{\frac{\delta}{4\ell}, \frac{1}{2}\}$. Suppose that the oracle returns an admissible solution (See Definition 2.3.1) for $T = \frac{2\rho\ln(m)}{\delta\epsilon}$ iterations in Algorithm 3, then for any constraint $1 \leq i \leq n$ we have: $(1-\epsilon)\sum_t \mathbf{M}(i, \mathbf{x}^t) \leq \delta T + \sum_t \mathbf{M}(\mathcal{D}^t, \mathbf{x}^t)$. Moreover $\tilde{\mathbf{x}}$ is a feasible solution of the Dual LP.*

2.3.2 A Framework for Fractional Packing or Covering Problems

Plotkin, Shmoys, and Tardos [84] presented a framework for solving fractional packing and fractional covering problems. A fractional packing problem can be written as follows:

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \lambda \mathbf{b} \\ & \mathbf{x} \in \mathcal{P} \end{aligned}$$

Algorithm 4 The framework for the fractional packing [84], with a slight modification.

1: Let $\lambda_0 = \max_i \mathbf{A}_i\mathbf{x}/\mathbf{b}_i$.

2: Let κ be a parameter with $\kappa \geq 4\lambda_0^{-1}\delta^{-1}\ln(2m\delta^{-1})$ and $\sigma = \frac{\delta}{4\kappa\rho}$.

3: **while** $\max_i \mathbf{A}_i\mathbf{x}/\mathbf{b}_i \geq (1-\delta)\lambda_0$ and $\mathbf{y}$ do not satisfy (P2) **do**

4: $\quad \mathbf{y}_i \leftarrow \frac{1}{\mathbf{b}_i}\exp(\kappa\mathbf{A}_i\mathbf{x}/\mathbf{b}_i)$ for all i.

5: $\quad$ Find $\tilde{\mathbf{x}} \in \mathcal{P}$ such that $\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} > \delta(\mathbf{y}^{\mathrm{T}}\mathbf{x} + \lambda\mathbf{y}^{\mathrm{T}}\mathbf{b})$.

6: $\quad \mathbf{x} \leftarrow (1-\sigma)\mathbf{x} + \sigma\tilde{\mathbf{x}}$

7: **end while**

where $\mathcal{P}$ is a convex polytope. The algorithm assumes that $\min_{x\in\mathcal{P}} \mathbf{c}^{\mathrm{T}}\mathbf{x}$ can be computed efficiently for all $\mathbf{c}$.

Definition 2.3.4. The **width parameter** of $\mathcal{P}$ is defined as $\rho = \max_i \max_{\mathbf{x}\in\mathcal{P}} \mathbf{A}_i\mathbf{x}/\mathbf{b}_i$. ($\mathbf{A}_i$ and $\mathbf{b}_i$ are the ith rows of $\mathbf{A}$ and $\mathbf{b}$ respectively. $\mathbf{A}_i\mathbf{x} \leq \lambda\mathbf{b}_i$ is the ith constraint of the packing problem.)

In Algorithm 4, we present the algorithm with a slightly modified description. The algorithm maintains a primal candidate $\mathbf{x}$ and computes a dual candidate $\mathbf{y}$ (using exponential weights). Then, the oracle solves a subproblem given $\mathbf{y}$(line 5) and returns the solution to the algorithm. The algorithm updates its primal candidate and proceeds to the next iteration. In this process, the framework is fixed but the oracle varies depending on the problem. So we need to design a problem-specific oracle in order to use the framework.

Definition 2.3.5. [84] Let λ^{OPT} be the optimal solution of the fractional packing problem. Let $\lambda = \max_i \mathbf{A}_i\mathbf{x}/\mathbf{b}_i$ and $C_P(\mathbf{y}) = \min_{\mathbf{x}\in\mathcal{P}} \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}$. $\mathbf{x}$ is δ-optimal if $\mathbf{x} \in \mathcal{P}$ and $\lambda \leq (1+\delta)\lambda^{OPT}$.

$$\text{(P1)} \qquad (1-\delta)\lambda\mathbf{y}^{\mathrm{T}}\mathbf{b} \leq \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}$$

$$\text{(P2)} \qquad \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} - C_P(\mathbf{y}) \leq \delta(\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \lambda\mathbf{y}^{\mathrm{T}}\mathbf{b})$$

Lemma 2.3.6 (A slight rewording of Lemma 2.3 in [84])**.** *If* $\mathbf{x}$ *and* $\mathbf{y}$ *satisfies (P1) and (P2) with* $\delta \leq 1/6$, $\mathbf{x}$ *is* 6δ*-optimal. Moreover, let* $\Phi = \mathbf{y}^{\mathrm{T}}\mathbf{b}$ *be a potential function and let* $\hat{\Phi}$ *be the potential function after the update. Then,* $\hat{\Phi} \leq (1-\kappa\sigma\delta\lambda)\Phi \leq (1-\Omega(\frac{\delta^2\lambda_0}{\rho}))\Phi$.

For any $\mathbf{x}$, its corresponding $\mathbf{y}$ in Algorithm 4 satisfies (P1). Therefore, if (P2) is also satisfied, then we have a proof that $\mathbf{x}$ is near-optimal. If on the other hand, (P2) is not satisfied then the algorithm shows that the potential drops significantly. Initially, $\Phi \leq m\exp(\kappa\lambda_0)$ and the algorithm terminates if $\Phi \leq \exp((1-\delta)\kappa\lambda_0)$. Combined with the above lemma, we obtain the following:

Theorem 2.3.7. *Given the initial solution with objective value* λ_0*, we find a solution with objective value* $(1-\delta)\lambda_0$ *in* $O(\frac{\kappa\rho}{\delta})$ *iterations.*

Theorem 2.3.7 holds even if we find $\tilde{\mathbf{x}}$ approximately, i.e., find $\tilde{\mathbf{x}}$ such that $\mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} \leq (1+\delta/2)C_P(\mathbf{y}^t) + (\delta/2)\lambda\mathbf{y}^{\mathrm{T}}\mathbf{b}$ [84]. It is also sufficient to compute $\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ and $\lambda\mathbf{y}^{\mathrm{T}}\mathbf{b}$ approximately (within a $1-\delta$ factor) based on the same ideas. Note that there is a slight difference between our parameters and the parameters in [84] which uses

the smallest possible κ, namely $\kappa = 4\lambda_0^{-1}\delta^{-1}\ln(2m\delta^{-1})$. We will use a larger value for κ in Chapter 7. This allows us to use structural properties of the polytope and subsequently allows us to approximate $\tilde{\mathbf{x}}$ easily.

Note that the admissibility condition in Section 2.3.1, i.e., the primal witness satisfying the constraints in expectation, implies $(P2)$ is not satisfied:

$$\mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} \leq (1+\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{b} \Longrightarrow (1+\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} \leq (1-\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \delta\mathbf{y}^{\mathrm{T}}\mathbf{b}/2$$

So it is sufficient to find a primal witness that satisfies the constraints in expectation like in the case of the multiplicative weights update method.

A Framework for Fractional Covering Problems

The framework for fractional covering problems is almost identical to the framework for fractional packing problems except two differences. First, we use $\mathbf{y} = \frac{1}{b_i}\exp(-\kappa\mathbf{A}_i\mathbf{x}/\mathbf{b}_i)$. Note that the exponent is negative rather than positive. Second, (P1) and (P2) conditions are replaced by the following conditions.

Definition 2.3.8. [84] Let $\lambda = \min_i \mathbf{A}_i\mathbf{x}/\mathbf{b}_i$ and $C_C(\mathbf{y}) = \min_{\mathbf{x}\in\mathcal{P}} \mathbf{y}^T\mathbf{A}\mathbf{x}$. $\mathbf{x}$ is δ-optimal if $\mathbf{x} \in \mathcal{P}$ and $\lambda \geq (1-\delta)\lambda^*$.

$$\text{(C1)} \qquad (1+\delta)\lambda\mathbf{y}^{\mathrm{T}}\mathbf{b} \geq \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}$$

$$\text{(C2)} \qquad C_C(\mathbf{y}) - \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} \leq \delta(\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \lambda\mathbf{y}^{\mathrm{T}}\mathbf{b})$$

The subroutine to find $\tilde{\mathbf{x}}$ is a maximization problem given the cost $\mathbf{y}$, i.e., $\mathrm{argmax}_{\mathbf{x}\in\mathcal{P}}\, \mathbf{y}^T\mathbf{A}\mathbf{x}$ or finding $\tilde{\mathbf{x}} \in \mathcal{P}$ such that $\mathbf{y}^T\mathbf{A}\tilde{\mathbf{x}} \geq (1+\delta)\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \delta\lambda\mathbf{y}^{\mathrm{T}}\mathbf{b}$. Again,

it is sufficient to find $\tilde{\mathbf{x}}$ that satisfies the constraints in expectation, i.e.,

$$\mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} \geq (1-\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{b} \Longrightarrow (1-\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{x}} \geq (1+\delta/2)\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \delta\mathbf{y}^{\mathrm{T}}\mathbf{b}/2$$

The rest of the algorithm and proofs is almost identical to the fractional packing framework. We have the following theorem.

Theorem 2.3.9. *[84] If the initial solution is 6ϵ-optimal for $\epsilon \leq 1/12$, we find a 3ϵ-optimal solution in $O(\epsilon^{-2}\rho\log(m\epsilon^{-1}))$ iterations.*

Mixing the Two Primal-Dual Algorithms

There are three main differences between the multiplicative weights update method in Section 2.3.1 and the fractional/packing covering framework in Section 2.3.2.

1. **Construction of Dual Candidates:**

2. **Polytope and Additional Constraints:**

3. **Admissibility:**

2.3.3 The SDP Feasibility Algorithm

Steurer's SDP feasibility algorithm [90] is based on the matrix multiplicative weights method given by Arora and Kale [12]. The feasibility algorithm is similar to the multiplicative weights update method [11]. The algorithm consists of multiple iterations of two-party game. In each iteration, the algorithm produces a candidate $\mathbf{X}$

(which corresponds to the weights of constraints in the multiplicative weights update method) and an oracle returns a separating hyperplane $(\mathbf{A}, b)$ (which corresponds to the dual witness). We define a canonical form of a semidefinite feasibility problem as follows:

Definition 2.3.10. For matrices $\mathbf{A}, \mathbf{B}$, let $\mathbf{A} \circ \mathbf{B}$ denote $\sum_{i,j} \mathbf{A}_{ij}\mathbf{B}_{ij}$. Let $\mathbf{A} \succeq 0$ mean that $\mathbf{A}$ is (positive) semidefinite and let $\mathbf{A} \succeq \mathbf{B}$ mean that $\mathbf{A} - \mathbf{B}$ is (positive) semidefinite. A **canonical form** of a semidefinite feasible problem can be written as:

$$\begin{aligned} ? \quad & \mathbf{C} \circ \mathbf{X} \geq \alpha \\ & \mathbf{A}_j \circ \mathbf{X} \leq b_j \quad \text{for all } j \\ & \mathbf{X} \succeq 0. \end{aligned}$$

$\mathbf{C} \circ \mathbf{X} \geq \alpha$ is a special constraint that corresponds to the (maximization) objective value of the SDP. We denote the set of the feasible solutions as $\mathcal{X}$.

In each iteration, an oracle of the feasibility algorithm is given a candidate $\mathbf{X}$ and does one of the following:

1. The algorithm declares the SDP feasible and returns a feasible (either fractional or integral) solution.

2. The algorithm returns a hyperplane $(\mathbf{A}, b)$ that separates $\mathbf{X}$ from a feasible area.

If the oracle returns a feasible solution, the algorithm stops and otherwise, it updates $\mathbf{X}$ and continues to the next iteration. If there exists a feasible solution, the

candidate $\mathbf{X}$ keep moving toward the feasible area and eventually the oracle cannot find a suitable separating hyperplane. Formally, we have the following definition and theorem.

Definition 2.3.11. [90] Let $\mathcal{X}$ be the set of feasible solutions of a SDP and let $(\mathbf{A}, b)$ be the hyperplane returned by the corresponding separation oracle (if it returns a hyperplane). A separation oracle is δ-separating and ρ-bounded if every hyperplane satisfy that

1. $\mathbf{A} \circ \mathbf{X} \leq b - \delta$ and $\mathbf{A} \circ \mathbf{X}' \geq b$ for all $\mathbf{X}' \in \mathcal{X}$, and

2. $-\rho\mathbf{D} \preceq \mathbf{A} - b\mathbf{D} \preceq \rho\mathbf{D}$.

δ and ρ correspond to the error parameter and the width parameter of the multiplicative weights update method.

Theorem 2.3.12. *[90] If $\mathcal{X}$ is non-empty, the oracle outputs a feasible (fractional or integral) solution within $T = O(\frac{\rho^2 \log n}{\delta^2})$ iterations.*

In order to implement the feasibility algorithm, the algorithm computes $\mathbf{X}$ approximately with the Johnson-Lindenstrauss lemma. Let d be the projected dimension of the Johnson-Lindenstrauss lemma, r be the degree of freedom (or precision), and T_M be the time for a multiplication between a linear combination of returned $\mathbf{A}$'s and a vector. Then $\mathbf{X}$ can be approximated within $O(d \cdot r \cdot T_M)$ time. For the purposes of this thesis, $d = O(\frac{\log n}{\delta^2})$, $r = O(\log \frac{1}{\delta})$, and $T_M = O(m')$.

2.4 The Matching Polytope

The first step in defining LPs for the matching problem and its variants is to find a proper relaxation of variables and constraints. In other words, we need to define fractional variables (in contrast to integral variables of the problem) and a polytope. There is well-known polytope for the matching.

Definition 2.4.1. Given a graph $G = (V, E)$ and a matching M, let $\mathbf{x}^M$ be an indicator vector of edges in M, i.e., x_{ij} is 1 if $(i, j) \in M$ and 0 otherwise. Then, the **matching polytope** of G is a convex hull of $\mathbf{x}^M$ for all matchings in G.

Theorem 2.4.2. *[86] For any graph $G = (V, E)$, the polytope defined by the following constraints is the matching polytope:*

$$\sum_{j:(i,j)\in E} x_{ij} \leq 1 \qquad \textit{for all } i \in V \qquad \textit{(vertex constraints)}$$

$$\sum_{i,j\in U} x_{ij} \leq \left\lfloor \frac{|U|}{2} \right\rfloor \qquad \textit{for all } U \subseteq V \qquad \textit{(set constraints)}$$

Observe that the feasibility of the constraint for a set U where $|U|$ is even follows from the feasibility of the vertex constraints, since each y_{ij} is summed up twice for $i, j \in U$. Therefore it suffices to focus on *odd* (size) sets U. Observe that the same observation holds for approximate feasibility, where the right hand sides of the equations corresponding to the constraints are multiplied by $(1 - \epsilon)$. From the definition of the matching polytope and Theorem 2.4.2, we can construct a linear program for the maximum matching problem with integrality gap one. LP1 and LP2 are the primal and dual linear programs for the maximum weighted matching problem. The

maximum cardinality matching problem is the case when $w_{ij} = 1$ for all $(i,j) \in E$ and have a similar linear program formulation.

$$\begin{array}{lll} \max & \sum_{i,j} w_{ij} x_{ij} & \\ \text{s.t} & \sum_j x_{ij} \leq 1 & \forall i \\ & \sum_{i,j \in U} x_{ij} \leq \left\lfloor \frac{|U|}{2} \right\rfloor & \forall U \\ & x_{ij} \geq 0 & \end{array} \quad \text{(LP1)}$$

$$\begin{array}{lll} \min & \sum_i y_i + \sum_U z_U & \\ \text{s.t} & y_i + y_j + \sum_{i,j \in U} z_U \geq w_{ij} & \forall (i,j) \in E \\ & y_i, z_U \geq 0 & \end{array} \quad \text{(LP2)}$$

The matching polytope is a well-studied object. One of the celebrated results on the matching polytope (with relation to the maximum matching problem) is the Cunningham-Marsh theorem.

Theorem 2.4.3 (The Cunningham-Marsh Theorem [31, 86])**.** *If all edge weights are integers, there exists an optimal solution for the maximum weighted matching linear program(LP1) such that all y_i and z_U are integers and $\{U | z_U > 0\}$ is laminar.*

Note that the theorem does **not** claim that y_i, z_U are $0/1$ – which is only true when $w_{ij} = 1$. This issue is important in solving the LP for the weighted matching case. The most accessible proof of laminarity that uses an adaptive two-stage optimization (See [86], volume A, pages 440–442). The proof finds an optimal solution that maximizes $\sum_U z_U |U|^2$ (among optimal solutions) and shows that such an optimal solution satisfies the laminarity. The integrality of the optimal solution follows from the laminarity.

Chapter 3

Graphs and Linear Measurements

Chapter Outline: In this chapter, we discuss three basic graph problems: connectivity, minimum spanning tree, and maximal matching. These problems demonstrate the difference between insertion-only streams and dynamic streams.

3.1 Connectivity (Spanning Forest)

In this section, we explain the algorithms for finding a spanning forest of a graph which is specified as a stream of updates over edges. We also discuss some applications of the algorithms: testing k-(edge-)connectivity, bipartiteness, and approximate minimum spanning tree.

In the insertion-only model, we have a trivial single-pass semi-streaming algorithm (Algorithm 5) for finding a spanning forest using a union-find data structure. The algorithm keeps a spanning forest of the input graph. As new edge arrives, if the edge

Algorithm 5 The SpanningForest Algorithm (for insertion-only streams)

```
Initialize a union-find data structure of size n and a forest F = ∅.
for each edge e = (u, v) do
    If find(u) = find(v), discard e. (u and v are already connected by the forest)
    Otherwise, F ← F ∪ {e} and union(u, v)
end for
Assert F as a spanning forest.
```

connects two disconnected component of the input graph, it is added to the spanning forest.

The algorithm for the insertion-only model is not applicable to dynamic streams because deletion of edges may disconnect a connected component. We now present a single-pass, semi-streaming algorithm that returns a spanning forest in a dynamic graph stream (Algorithm 6). The algorithm is based on constructing sketches for ℓ_0-sampling from the rows of a matrix that we now define.

Definition 3.1.1. Given an unweighted graph $G = (V, E)$, define the $n \times \binom{n}{2}$ matrix A_G with entry $(i, (j, k)) \in [n] \times \binom{[n]}{2}$ defined by

$$a_{i,(j,k)} = \begin{cases} 1 & \text{if } i = j \text{ and } (j,k) \in E \\ -1 & \text{if } i = k \text{ and } (j,k) \in E \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{a}_1, \ldots, \mathbf{a}_n$ be the rows of A_G where $\mathbf{a}_i$ corresponds to a vertex i. The following preliminary lemma follows immediately from the above definition.

Algorithm 6 The SpanningForest Algorithm (for dynamic streams)

1: Sketch $\mathbf{a}_1, \ldots, \mathbf{a}_n$ using the sketch matrices $\mathcal{S}_1, \ldots, \mathcal{S}_t$ where $t = O(\log n)$.

2: Initialize the set of supervertices as $\hat{V} = V$.

3: **for** $r \in [t]$ **do**

4: For each $s \in \hat{V}$, try to sample an inter-supervertex edge using the sketch $\sum_{i \in s} \mathcal{S}_r(a_i)$

5: Update $\hat{V}$ by collapsing connected supervertices.

6: **end for**

7: Assert that G has $|\hat{V}|$ connected components and that any maximal acyclic subgraph of the set of sampled edges is a spanning forest.

Lemma 3.1.2. *Let $E_S = E(S, V \setminus S)$ be the set of edges across the cut $(S, V \setminus S)$. Then, $|E_S| = \ell_0(\mathbf{x})$ where $\mathbf{x} = \sum_{i \in S} \mathbf{a}_i$. Furthermore, $\mathbf{x} \in \{-1, 0, 1\}^{\binom{n}{2}}$ with $|\mathbf{x}_{(j,k)}| = 1$ iff $(j, k) \in E_S$.*

Our algorithm is based on the following simple $O(\log n)$ stage process. In the first stage, we find an edge incident on each vertex. We then collapse each of the resulting connected components into a "supervertex". In each subsequent stage we find an edge from each supervertex to another supervertex if one exists. If the graph has $cc(G)$ connected components, the difference between the number of supervertices and $cc(G)$ halves with each stage and therefore after $O(\log n)$ stages the graph has collapsed into $cc(G)$ supervertices and will not collapse further. The challenge is to emulate the algorithm space-efficiently in a single pass over a dynamic graph stream.

To emulate the algorithm efficiently, we will construct a number of sketches of each $\mathbf{a}_i$ that will facilitate the ℓ_0-sampling. Let the sketch matrices be $\mathcal{S}_1, \ldots, \mathcal{S}_t$ where $t = O(\log n)$. Constructing a sketch of each vertex, using each sketch matrix requires $O(nt \log^2 n)$ for constant δ. For our algorithm it will suffice to set the ℓ_0-sampling parameters as $\epsilon = 0$ and $\delta = 1/100$.

To sample an edge incident on i, we can use the sketch $\mathcal{S}_1(\mathbf{a}_i)$. Appealing to Lemma 3.1.2, this succeeds with probability $1 - \delta$. At the next step, to sample an edge from the supervertex $s = \{i, j\}$ we can use the sketch $\mathcal{S}_2(\mathbf{a}_i) + \mathcal{S}_2(\mathbf{a}_j)$ to find such an edge with probability at least $1 - \delta$. This follows by the linearity of the sketches and by appealing again to Lemma 3.1.2. We emphasize that using $\mathcal{S}_1(\mathbf{a}_i) + \mathcal{S}_1(\mathbf{a}_j)$ rather than $\mathcal{S}_2(\mathbf{a}_i) + \mathcal{S}_2(\mathbf{a}_j)$ will *not* work for two reasons: (a) it would mean that we has used $\mathcal{S}_1$ in the first stage to determine our use of $\mathcal{S}_1$ in the second stage and such adaptivity is not permissible in general and (b) the probabilities from Lemma 3.1.2 would not be independent. In short, we need to use a new sketching matrix $\mathcal{S}_i$ in each round. See Algorithm 6 for the full algorithm.

The correctness of the algorithm follows because in each stage we expect to decrease $|\hat{V}| - cc(G)$ by at least a third by appealing to the linearity of expectation. Hence, it suffices to set $t = O(\log n)$.

Theorem 3.1.3. *There exists a single-pass, $O(n \cdot \log^3 n)$-space, $O(m \operatorname{polylog} n)$-time algorithm for dynamic connectivity. The algorithm returns a spanning forest of the graph.*

3.1.1 k-Edge-Connectivity

In this section, we present single-pass algorithms for k-edge-connectivity. The starting point for the algorithm is the following simple k phase algorithm:

1. For $i = 1$ to k: Let F_i be a spanning forest of $(V, E \setminus \bigcup_{j=1}^{i-1} F_j)$

2. Then $(V, F_1 \cup F_2 \cup \ldots \cup F_k)$ is k-edge-connected iff $G = (V, E)$ is at least k-edge-connected.

The correctness of this algorithm is simple to show.

Lemma 3.1.4. *Given a graph $G = (V, E)$, for $i \in [k]$, let F_i to be a spanning forest of $(V, E \setminus \bigcup_{j=1}^{i-1} F_j)$. $(V, F_1 \cup F_2 \cup \ldots \cup F_k)$ is k-edge-connected iff $G = (V, E)$ is at least k-edge-connected.*

Proof. Let $E' = F_1 \cup \ldots \cup F_k$. Consider a cut $(S, V \setminus S)$ and let $E_S \subset E$ be the set of edges that cross this cut. Similarly, let E'_S be the set of edges among E' that cross this cut. Clearly $|E_S| \geq |E'_S|$ since $E' \subset E$. Hence, it suffices to prove that $|E'_S| \geq k$ if G is k-edge-connected. Suppose there exists i such that $F_i \cap E_S = \emptyset$ (otherwise we are done by the edge-disjointness of the k spanning forests.) Then $E_S \cap (F_1 \cup F_2 \cup \ldots \cup F_{i-1}) = E_S$ and hence $|E'_S| = |E_S| \geq k$. $\square$

We call $F_1, \cdots, F_k$ a *witness* of k-connectivity. Both algorithms for insertion-only streams and dynamic streams return a witness. Such witness will be used for the graph sparsification algorithm (see Chapter 5).

Algorithm 7 The k-CONNECTIVITY Algorithm (for insertion-only streams)

1: Initialize union-find data structures $UF_1, UF_2, \cdots, UF_k$ of size n and forests $F_1, F_2, \cdots, F_k = \emptyset$.

2: **for** each edge $e = (u, v)$ **do**

3: Find the first i such that $UF_i.\text{find}(u) \neq UF_i.\text{find}(v)$. (If $UF_i.find(u) = UF_i.find(v)$ for all i, discard e).

4: $F_i \leftarrow F_i \cup \{e\}$ and $UF_i.\text{union}(u, v)$.

5: **end for**

6: Assert $F_1, F_2, \cdots, F_k$ as a witness.

The algorithm for insertion-only streams is rather straight-forward (Algorithm 7). Since we know which edge will be selected for the output spanning forest, we can (virtually) create a new stream without those edges belong to the spanning forest. Repeating k times, we obtain a single-pass, semi-streaming algorithm.

The algorithm for dynamic streams builds upon ideas in the previous section. It is relatively straight-forward to design a $O(k)$-pass algorithm using the spanning forest algorithm from the previous section. However, by exploiting the linearity of sketches, we show that it is possible to test k-edge connectivity in only one pass (Algorithm 8).

Because of instantiations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ are independent, we may update them as claimed.

Theorem 3.1.5. *There exists a one-pass, $O(kn)$-space, $O(km\alpha(n))$-time algorithm for testing k-connectivity in insertion-only graph streams. There exists a one-pass,*

Algorithm 8 The k-CONNECTIVITY Algorithm (for dynamic streams)

0: Construct the sketches for k independent instantiations $\mathcal{I}_1, \ldots, \mathcal{I}_k$ of the spanning forest algorithm.

0: For $i \in [k]$:

1. Use the sketches for $\mathcal{I}_i$ to find a spanning forest F_i of $(V, E \setminus F_1 \cup \ldots \cup F_{i-1})$
2. Update the sketches for $\mathcal{I}_{i+1}, \ldots, \mathcal{I}_k$ by deleting all edges in F_i

0: Assert $F_1, F_2, \cdots, F_k$ as a witness.

$O(kn \log^3 n)$-space, $O(km \operatorname{polylog} n)$-time algorithm for testing k-connectivity in dynamic graph streams.

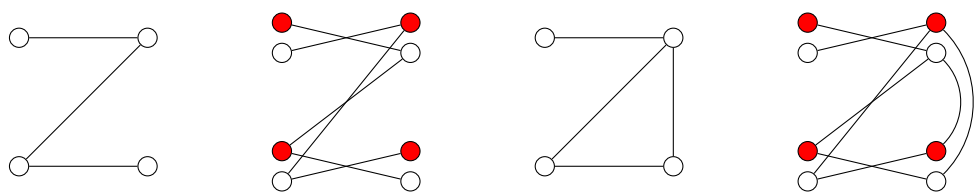
3.1.2 Bipartiteness

Next, we reduce the bipartiteness problem to the problem of counting the number of connected components. The reduction is based on the following local mapping.

Definition 3.1.6. For a graph $G = (V, E)$, let $D(G) = (V', E')$ be the graph constructed as follows. For each $v \in V$ construct $v_1, v_2 \in V'$ and for each edge $(u, v) \in E$, create two edges (u_1, v_2) and (u_2, v_1). $D(G)$ is called the *bipartite double cover* of G.

See Figure 3.1 for an illustration of the action of D.

Lemma 3.1.7. *Let K be the number of connected components in G. Then, $D(G)$ has $2K$ connected components if and only if G is bipartite.*



(a) G (b) $D(G)$ (c) G' (d) $D(G')$

Figure 3.1: The map D doubles the number of connected components iff the graph is bipartite.

Proof. Let $G_1, G_2, \cdots, G_K$ be the connected components in G. By the construction, $D(G)$ consists of $D(G_1)$, $D(G_2)$, $\cdots$, $D(G_K)$. If we sum the number of connected components in each $D(G_i)$, we have the total number of connected components in $D(G)$.

Suppose that there exists an odd cycle in G_i which contains u. The odd cycle corresponds to a path from u_1 to u_2 or vice versa in $D(G_i)$. And each path from u to v in G_i corresponds to a path from u_1 to v_1 or v_2 and a path from u_2 to the other copy of v. Since G_i is connected, there exists a path from u to any $v \in V$. So there is a path from u_1 to any vertex in $v \in V'$ and $D(G_i)$ is connected. On the other hand, if G_i is bipartite, $D(G_i)$ is not connected because any path from u_1 to u_2 in G' corresponds to an odd cycle in G.

If G is bipartite, every G_i is bipartite. Therefore, there are $2K$ connected components in $D(G)$. On the other hand, if G is not bipartite, there must be G_i that is not bipartite. So there is less than $2K$ connected components. $\square$

Applying Theorem 3.1.3 on G and $D(G)$, we can compute the number of connectivity components in G and $D(G)$. Combining with the above lemma, we obtain the following result:

Theorem 3.1.8. *There exists a one-pass, $O(n)$-space, $O(m\alpha(n))$-time algorithm for testing bipartiteness in insertion-only graph streams. There exists a one-pass, $O(n \log^3 n)$-space, $O(m \operatorname{polylog} n)$-time algorithm for testing bipartiteness in dynamic graph streams.*

3.2 Minimum Spanning Tree (MST)

In this section, we present algorithms for finding exact and approximate MST. In the insertion-only model, we have a single-pass, semi-streaming algorithm for finding an exact MST. However, it takes multiple passes to find an exact MST in the dynamic model.

3.2.1 Exact Minimum Spanning Tree

We assume that there are no two edges with the same edge weight. We achieve by tie-breaking edge weight arbitrarily (for example, using lexicographic order when two edges have the same edge weight). Then, there is a unique MST and the following lemma and the algorithm for the exact MST (Algorithm 9) follows from Prim's algorithm.

Lemma 3.2.1. *Let T_1 be the MST of G_1. Then, the MST of $T_1 \cup G_2$ is the MST of $G_1 \cup G_2$.*

Algorithm 9 The MST Algorithm (for insertion-only streams)

```
Initialize a spanning tree T = ∅.
repeat
    Read n edges from the stream. Let H be the set of edges.
    Compute the minimum spanning tree of T ∪ H.
    Remember the new minimum spanning tree as T.
until the end of the stream
Assert T as the minimum spanning tree.
```

Now we proceed to the algorithm for dynamic streams. Our starting point is Boruvka's algorithm for finding the MST. This algorithm proceeds in $O(\log n)$ phases. In each phase, the minimum weight edge incident on each vertex is added and the resulting connected components are collapsed to form new vertices. This algorithm can be implemented in the dynamic stream setting in $O(\log^2 n)$ passes by emulating each phase in $O(\log n)$ passes of the dynamic graph stream. We emulate a phase as follows: In the first pass, we ℓ_0-sample an incident edge on each vertex without considering the weights. Suppose we sample an edge with weight w_v on vertex v. In the next pass, we again ℓ_0-sample incident edges but this time we ignore all edges of weight at least w_v on vertex v when we construct the sketch. Repeating this process $O(\log n)$ ensures that we succeed in finding the minimum weight edge incident on

each vertex. Thus the algorithm takes $O(\log^2 n)$ passes as claimed. In the rest of this section, we transform the algorithm into a new algorithm that uses $O(n^{1+1/p})$ space and $O(p)$ passes which translates to a $O(\log n/\log\log n)$-pass, $O(n \operatorname{polylog} n)$-space algorithm.

Reducing the Number of Passes.

Our first step is to show that $O(\log n \log\log n)$ passes suffice. The algorithm is based on the observation that the number of vertices under consideration in the ith phase is at most $n/2^i$. Hence, during the ith phase we can afford to sample $t_i = 2^i$ incident edges without violating the semi-streaming space restriction. Therefore, the ith phase can be emulated in $O(\log_{t_i} n)$ passes which implies that the total number of passes is $\sum_{i=1}^{\log n} \log_{2^i} n = O(\log n \log\log n)$.

The next step is to reduce the number of phases. The basic idea is to not just find the lightest incident edge for each vertex, but to find the k lightest edges. It follows from the next lemma that this allows us to reduce the number of vertices by a factor k.

Lemma 3.2.2. *In a simple weighted graph $G = (V, E)$, if $E' \subset E$ contains the k lightest incident edges on each vertex, then we can identify the lightest edge in the cut $(S, V \setminus S)$ for any subset $S \subset V$ of size at most k.*

Proof. If $|S| \leq k$ then we know the lightest edge incident on each $v \in S$ in $E(S, V \setminus S)$ because v has at most $k-1$ neighbors in S. □

Unfortunately, after the first phase the graph under consideration is a multi-graph and the above lemma does not apply directly. For example, the lightest k incident edges on v may all connect v to the same neighbor. However, we can rectify this situation by constructing $O(\log n)$ random partitions $P_1, \ldots, P_{O(\log n)}$ of the vertices where each partition is of size $2k$. For each vertex v and each partition $P = \{V_1, \ldots, V_{2k}\}$ we find the lightest edge from v to each V_i. Let E' be the set of edges collected.

Lemma 3.2.3. *With high probability, E' contains the k lightest edges to distinct neighbors.*

Proof. Let N_v be the k closest neighbors of v and consider $u \in N_v$. With high probability there exists a partition $P = \{V_1, \ldots, V_{2k}\}$ where u is the only element in $V_i \cap N_v$ for some i. Hence, we identify the lightest edge between u and v. □

The next theorem is proved by carefully combining the above idea with the $O(\log n \log\log n)$ pass algorithm.

Theorem 3.2.4. *There exists a $O(p)$-pass, $\tilde{O}(n^{1+1/p})$-space algorithm that finds the MST of a dynamic graph stream. In particular there is a semi-streaming algorithm that uses $O(\log n/\log\log n)$ passes.*

Proof. Suppose at some point we have $n^{1+1/p}/t$ remaining vertices. Then we can find the $\sqrt{t}$ closest neighbors of a vertex in $O(\log_{\sqrt{t}} n)$ passes as follows: construct $O(\log n)$ random partitions each of size $2\sqrt{t}$ and then use $O(\log_{\sqrt{t}} n)$ successive batches of $\sqrt{t}$ sketches for ℓ_0-sampling to find the lightest edges between each vertex and a vertex

in each set of each partition. Let n_i be the number of vertices at the start of the ith phase and define $t_i = n^{1+1/p}/n_i$. Then $t_1 = n^{1/p}, t_2 = n^{3/(2p)}, t_3 = n^{9/(4p)} \ldots$ and hence $\sum_i \log_{t_i}(n) = O(p)$. $\square$

3.2.2 Approximate Minimum Spanning Trees

Unless we have $\Omega(n^2)$ space, it is not possible to compute the exact minimum spanning tree in dynamic streams. However, we can $(1+\epsilon)$-approximate the MST in one-pass with near linear space. We reduce the problem of estimating the weight of the MST to the problem of counting the number of connected components in graphs. The reduction uses an idea due to Chazelle et al. [27]. Consider a graph G with edge weights are in the range $[1, W]$ where $W = \text{poly}(n)$. We will assume that G is connected but our algorithm can be used to estimate the weight of the minimum weight spanning forest if G is unconnected. Let G_i be the subgraph of G consisting of all edges whose weight is at most $w_i = (1+\epsilon)^i$ and let $cc(H)$ denote the number of connected components of a graph H.

Lemma 3.2.5. *Let T be a minimum spanning tree of G and set $r = \lceil \log_{1+\epsilon} W \rceil$. Then*

$$w(T) \leq n - (1+\epsilon)^r + \sum_{i=0}^{r} \lambda_i cc(G_i) \leq (1+\epsilon) w(T)$$

where $\lambda_i = (1+\epsilon)^{i+1} - (1+\epsilon)^i$.

Proof. Consider the G' formed by rounding each edge weight up to the nearest power of $(1+\epsilon)$. Then it is clear that $w(T) \leq w(T') \leq (1+\epsilon) w(T)$ where T' is a minimum

spanning tree of G'. It remains to compute $w(T')$ and we do this by considering the operation of Kruskal's algorithm on G'. Kruskal's algorithm will first add $n - cc(G_0)$ edges of weight 1, then $cc(G_0) - cc(G_1)$ edges of weight $(1+\epsilon)$ etc. The total weight of edges added will be

$$w(T') = (n - cc(G_0)) + \sum_{i=1}^{r-1} (1+\epsilon)^i (cc(G_i) - cc(G_{i+1}))$$

which simplifies to give the claimed quantity. □

Hence, we can estimate the weight of the minimum spanning tree using the connectivity algorithm.

Theorem 3.2.6. *There exists a single-pass, $O(\epsilon^{-1} \cdot n \cdot \log^4 n)$-space, $O(m \cdot \text{polylog}\, n)$-time algorithm that $(1+\epsilon)$ approximates the weight of the minimum spanning tree of a dynamic graph.*

3.3 Maximal Matching

The maximal matching in the insertion-only model is straight-forward (Algorithm 10). The algorithm finds a maximal matching in one pass, $O(n)$ space, and $O(m)$ time.

To find a maximal matching in dynamic streams we appeal to a result by Lattanzi et al. [70] for finding a maximal matching (with no deletions) in the MapReduce model. Their algorithm repeatedly samples a subset of $\eta = O(n^{1+1/p})$ edges in each

Algorithm 10 The MaximalMatching Algorithm (for insertion-only streams)

1: $\mathcal{M} \leftarrow \emptyset$.

2: **for** each edge $e = (u, v)$ **do**

3: If u or v is matched in $\mathcal{M}$, discard e.

4: Otherwise, $\mathcal{M} \leftarrow \mathcal{M} \cup \{e\}$.

5: **end for**

6: Return $\mathcal{M}$.

Algorithm 11 The MaximalMatching Algorithm (for dynamic streams)

1: $\mathcal{M} \leftarrow \emptyset$.

2: **for** $O(p)$ rounds **do**

3: Sample $n^{1+1/p}$ edges uniformly at random.

4: Find a maximal matching among the sampled edges and include them in $\mathcal{M}$

5: Exclude all matched vertices and edges incident to them.

6: **end for**

7: Return $\mathcal{M}$.

round and finds a maximal matching among the sampled edges, for p rounds; the vertices in the matching are excluded in the subsequent rounds. The same algorithm can be implemented in the dynamic semi-streaming model using sketches for ℓ_0-sampling (see Algorithm 11). In each pass, we can sample η edges uniformly at random and hence, each round of the algorithm corresponds to a single pass. Summarizing, we obtain the following theorem:

Theorem 3.3.1. *There exists a $O(p)$-pass, $O(n^{1+1/p})$-space algorithm that finds a*

maximal matching in the dynamic graph streams with high probability.

Chapter 4

Distance Spanners

In this section we discuss the construction of distance preserving synopsis structures — that is given a graph G, we want to create a small-space representation such that we can answer queries such as the distance from between two specified vertices. Since the synopsis may not be able to answer the distance exactly, we will be interested in the distortion of the answers. In particular, an α-spanner is a spanner that answers all the distance queries within an approximation factor of $\alpha > 1$.

4.1 A Space-efficient Local BFS Algorithm

There has been several papers that investigate the construction of spanners in the (insertion only) streaming model [44, 17], and the best result is a $(2k-1)$-spanner using $O(n^{1+1/k})$ space in a single pass. Moreover it is also known that this trade-off of approximation factor and space is optimal. These algorithms use local breadth-

first-search trees (BFS); and the roots of these trees are connected to every vertex in the other tree. However, as intermediate steps, the algorithm also maintains BFS trees from other vertices such that the entire algorithm can proceed in a single pass [17]. However that algorithm cannot be implemented in the presence of deletions. We first discuss a simple adaptation of that algorithm using ℓ_0-sampling that can be implemented in $O(k)$ passes.

Algorithm ℓ_0-BFS: The algorithm has two parts.

- **Part 1: Growing Trees** This part consists of $k-1$ phases where at the end of phase i we have constructed a set of rooted vertex-disjoint trees $T_i[v]$ where v is the root of the tree and the set of roots is going to be denoted by S_i. Each $T_i[v]$ will have the property that the distance between a leaf and v is at most i. At the end of phase i there may also be many vertices that are not in a tree.

  - *First phase*: Pick each vertex with probability $n^{-1/k}$. Call the selected vertices S_1. We will start growing trees around the selected vertices where the selected vertices will be the roots of their respective trees. Specifically, if vertex u is adjacent to a selected vertex v add (u, v) to the tree $T_1[v]$. If u is adjacent to multiple selected vertex, add (u, v) to one of the trees arbitrarily. If a vertex u is not adjacent to any selected vertex, we remember the set of incident edges $L(u)$.

- i^{th} *phase* Construct S_i from $S_{(i-1)}$ by sampling each vertex with probability $n^{-1/k}$. For each $v \in S_i$ initialize $T_i[v] = T_{(i-1)}[v]$. If u is adjacent to a vertex w in some tree $T_i[v]$ add (u, w) to $T_i[v]$. If u is adjacent to multiple trees, just add u to one of the trees (doesn't matter which). Again if a vertex is not adjacent to any selected tree, then remember the set of incident edges L(u) where you only store one edge to vertices in the same $T_{(i-1)}$ tree.

- **Part 2: Final Clean Up:** Once we've defined $T_{(k-1)}[v]$ for $v \in S_{(k-1)}$ (and deleted all vertices not in these trees) let V' be the set of vertices involved in the $T_{(k-1)}$ trees (and roots, leaves, intermediate vertices etc.) For each $u \in V'$ add (at most one) edge to a vertex in some $T_{(k-1)}[v]$ if such an edge exists.

Following the proofs outlined in [44, 17], we can show that:

Theorem 4.1.1. *Algorithm ℓ_0-BFS uses $O(k)$ passes and gives a $(2k-1)$-spanner using $\tilde{O}(n^{1+1/k})$ space.*

4.2 A Pass-efficient Recursive Contraction Algorithm

The BFS growth algorithm gives an optimum trade-off between space $\tilde{O}(n^{1+1/k})$ and approximation $2k-1$, but uses $O(k)$ passes which is less desirable. For example, to achieve a semi-streaming space bound, the number of passes is $O(\log n)$. While this

is interesting[1], a natural question arises: can we produce a spanner in fewer passes? In what follows, we answer the question in the affirmative and provide an algorithm that uses $O(\log k)$ passes at the expense of a worse approximation factor.

The main intuition for the improvement in number of passes is the following: in the BFS algorithm we are growing regions of small diameter (at various granularities) and in each pass we are growing the region by 1 edge. Thus the growth of the regions is slow. Moreover in each of these steps we are using $O(n)$ space (if the graph is dense) — yet the space allowed for the vertex is $\tilde{O}(n^{1+1/k})$ and we expect the extra space to matter precisely when the graphs are dense! But if we are growing BFS trees, the extra edges are simply not useful. We will therefore relax the BFS constraint — this will allow us to grow the regions faster. The algorithm RecurseConnect is presented below.

Algorithm RecurseConnect **:**

1. The algorithm proceeds in phases which correspond to the passes. In the phase i, we have a graph $\tilde{G}_i$ which is contraction of the graph $G = \tilde{G}_0$; that is, subsets of vertices of the G have been merged into a collection of supervertices. This process will proceed recursively; and we will maintain $|\tilde{G}_i| \leq n^{1-(2^i-1)/k}$. Therefore after $\log k$ passes we have a graph of size $\sqrt{n}$ and we remember the connectivity between every pair of vertices in $O(n)$ space. We describe how to achieve $\tilde{G}_{i+1}$ from $\tilde{G}_i$ below.

[1]Observe that we cannot "sort" a stream in $O(\log n)$ passes in a read only model.

2. For each vertex in $\tilde{G}_i$ we sample $n^{2^i/k}$ distinct neighbors — vertices p, q in $\tilde{G}_i$ (note that p, q are subsets of the original vertex set) are neighbors in $\tilde{G}_i$ if there exists an edge $(u, v) \in G$ such that $u \in p$ and $v \in q$. To do this, for each vertex in $\tilde{G}_i$, we independently partition the vertex set of $\tilde{G}_i$ into $\tilde{O}(n^{2^i/k})$ subsets, and use an ℓ_0 sampler for each partition. This can be achieved in $\tilde{O}(n^{1/k})$ space per vertex and in total $\tilde{O}(n^{1+1/k})$ space, using the hypotheses $|\tilde{G}_i| \leq n^{1-(2^i-1)/k}$. Note that using the standard coupon-collector argument we can find all the vertices in $\tilde{G}_i$ whose degree is at most $n^{2^i/k}$ as well.

3. The set of sampled edges in $\tilde{G}_i$ gives us a graph H_i. We now choose a clustering of H_i where the centers of the clusters are denoted by C_i. Consider the subset S_i of vertices of H_i which have degree at least $n^{2^i/k}$ — we will ensure that C_i is a maximal (*not* maximum) subset of S_i which is independent in H_i^2. This is a standard construction used for the approximate k-center problem. More specifically: We start from the set C_i^0 being an arbitrary vertex in H_i. We repeatedly augment C_i^j to C_i^{j+1} by adding vertices which are (i) at distance at least 3 (as measured in number of hops in H_i) from each vertex in C_i^j. and (ii) have degree at least $n^{2^i/k}$. Denote the final C_i^j, when we cannot add any more vertices, as C_i. Observe that $|C_i| \leq |\tilde{G}_i|/n^{2^i/k} \leq n^{1-(2^{(i+1)}-1)/k}$.

4. For each vertex $p \in C_i$ all neighbors of p in H_i are assigned to p. For each vertex q with degree at least $n^{2^i/k}$ in $\tilde{G}_i$, if it is not chosen in C_i, we have a center p in C_i within 2 hops of q in H_i; then q is assigned to p as well.

5. We now collapse all the vertices assigned to $p \in C_i$ into a single vertex and these $|C_i|$ vertices define $\tilde{G}_{i+1}$.

It is immediate that after at most $\log k$ passes $\tilde{G}_i$ shrinks below $\sqrt{n}$. We now prove the approximation achieved by the above algorithm. *Observe that the distances are now primarily in the collapsed vertices.*

Lemma 4.2.1. *The distance between any adjacent $u, v \in G$ is at most $5^{\log k} - 1 = k^{\log_2 5} - 1$.*

Proof. Define the maximum distance between any u, v which are in the same collapsed set in $\tilde{G}_i$ as a_i. Note that $a_1 = 4$ since the clustering C_1 has radius 2, and therefore any collapsed pair are at a distance at most 4. For $i > 1$ observe that $a_{i+1} = 5a_i + 4$ and the result follows. □

Theorem 4.2.2. RecurseConnect *gives us a $k^{\log_2 5} - 1$ approximation using $\log k$ passes and $\tilde{O}(n^{1+1/k})$ space.*

Chapter 5

Graph Sparsification

Chapter Outline: In this chapter, we present the semi-streaming graph sparsification algorithm. We first discuss the minimum cut algorithm to demonstrate the high-level idea and then apply the same idea to the graph sparsification. Both algorithms require an access to random bits for edges which are not available for the insertion-only model.

5.1 A Warmup: The Minimum Cut Problem

To warm up, we start with a one-pass semi-streaming algorithm (Algorithm 12, for the minimum cut problem. This will introduce some the ideas used in the subsequent sections on sparsification. The algorithm is based on Karger's Uniform Sampling Lemma (Lemma 2.2.1) [67].

See Algorithm 12 for our minimum cut algorithm. The algorithm generates a

Algorithm 12 The MinCut Algorithm. Steps 1-5 are performed together in a single pass. Step 6 is performed in post-processing.

1: Let $h_i : E \to \{0, 1\}$ be a uniform hash function for $i \in \{1, \ldots, \lfloor 2 \log n \rfloor\}$.

2: **for** $i = 0$ **to** $\lfloor 2 \log n \rfloor$ **in parallel do**

3: Let G_i be the subgraph of G containing edges e such that $\prod_{j \leq i} h_j(e) = 1$ ($G_0 = G$).

4: Let $H_i \leftarrow k$-Connectivity (G_i) (Algorithm 7 for the insertion-only model or Algorithm 8 for the dynamic model) for $k = O(\epsilon^{-2} \log n)$

5: **end for**

6: **return** $2^j \lambda(H_j)$ where $j = \min\{i : \lambda(H_i) < k\}$

sequence of graphs $G = G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ where G_i is formed by independently removing each edge in G_{i-1} with probability 1/2. Simultaneously we use k-Connectivity to construct a sequence of graphs $H_0, H_1, H_2, \ldots$ where H_i contains all edges in G_i that participate in a cut of size k or less. The idea is that if i is not too large, $\lambda(G)$ can be approximated via $\lambda(G_i)$ and if $\lambda(G_i) \leq k$ then $\lambda(G_i)$ can be calculated from H_i.

Theorem 5.1.1. *Assuming access to fully independent random hash functions, there exists a single-pass, $O(\epsilon^{-2} n \log^4 n)$-space algorithm that $(1+\epsilon)$-approximates the minimum cut in the dynamic graph stream model. The requirement of fully independent random hash functions will be eliminated in Theorem 5.4.2*

Proof. If a cut in G_i has less than k edges that cross the cut, the witness contains all

such edges. On the other hand, if a cut value is larger than k, the witness contains at least k edges that cross the cut. Therefore, if G_i is not k-edge-connected, we can correctly find a minimum cut in G_i using the corresponding witness.

Let $\lambda(G)$ be the minimum cut size of G and let

$$i^* = \left\lfloor \log \max \left\{ 1, \frac{\lambda \epsilon^2}{6 \log n} \right\} \right\rfloor .$$

For $i \leq i^*$, the edge weights in G_i are all 2^i and therefore G_i approximates all the cut values in G w.h.p. by Lemma 2.2.1. Therefore, if MinCut returns a minimum cut from G_i with $i \leq i^*$, the returned cut is a $(1+\epsilon)$-approximation.

By Chernoff bound, the number of edges in G_{i^*} that crosses the minimum cut of G is $O(\epsilon^{-2} \log n) \leq k$ with high probability. Hence, MinCut terminates at $i \leq i^*$ and returns a $(1+\epsilon)$-approximation minimum cut with high probability. □

5.2 A Simple Sparsification

See Algorithm 13 for a simple sparsification algorithm. The algorithm extends the MinCut Algorithm by taking into account the connectivity of different edges.

Lemma 5.2.1. *Assuming access to fully independent random hash functions,* Simple-Sparsification *uses $O(\epsilon^{-2} n \log^5 n)$ space and the number of edges in the sparsification is $O(\epsilon^{-2} n \log^3 n)$.*

Proof. Each of the $O(\log n)$ instance of k-Connectivity runs in $O(kn \log^2 n)$ space. Hence, the total space used by the algorithm is $O(\epsilon^{-2} n \log^5 n)$. Since the total number

Algorithm 13 The SIMPLE-SPARSIFICATION Algorithm. Steps 1-5 are performed in a single pass. Step 6 is performed in post-processing.

1: Let $h_i : E \to \{0, 1\}$ be a uniform hash function for $i \in \{1, \ldots, \lfloor 2 \log n \rfloor\}$.

2: **for** $i = 0$ **to** $\lfloor 2 \log n \rfloor$ **in parallel do**

3: Let G_i be the subgraph of G containing edges e such that $\prod_{j \leq i} h_j(e) = 1$ ($G_0 = G$).

4: Let $H_i \leftarrow k$-CONNECTIVITY (G_i) (Algorithm 7 for the insertion-only model or Algorithm 8 for the dynamic model) for $k = O(\epsilon^{-2} \log n)$

5: **end for**

6: For each edge $e = (u, v)$, find $j = \min\{i : \lambda_e(H_i) < k\}$. If $e \in H_j$, add e to the sparsification with weight 2^j. Return the constructed sparsification.

of edges returned is $O(kn \log n)$, the number of edges in the sparsification is also bounded by $O(\epsilon^{-2} n \log^3 n)$. □

As mentioned earlier, the analysis of our sparsification result uses a modification of Theorem 2.2.3 that arises from the fact that we will not be able to independently sample each edge. The proof of Theorem 2.2.3 is based on the following version of the Chernoff bound.

Lemma 5.2.2 (Fung et al. [46])**.** *Consider any subset C of edges of unweighted edges, where each edge $e \in C$ is sampled independently with probability p_e for some $p_e \in (0, 1]$ and given weight $1/p_e$ if selected in the sample. Let the random variable X_e denote the weight of edges e in the sample; if e is not selected, then $X_e = 0$. Then, for any*

$p \leq p_e$ for all edges e, any $\epsilon \in (0,1]$, and any $N \geq |C|$, the following bound holds:

$$\mathbb{P}\left[\left|\sum_{e \in C} X_e - |C|\right| \geq \epsilon N\right] < 2\exp(-0.38\epsilon^2 pN) \ .$$

We will need to prove an analogous lemma for our sampling procedure. Consider the Simple-Sparsification algorithm as a sampling process that determines the edge weight in the sparsification. Initially, the edge weights are all 1. For each round $i = 1, 2, \ldots$ if an edge e is not k-connected in G_{i-1}, we freeze the edge weight. For an edges e that is not frozen, we sample the edge with probability $1/2$. If the edge is sampled, we double the edge weight and otherwise, we assign weight 0 to the edge.

Definition 5.2.3. Let $X_{e,i}$ be random variables that represent the edge weight of e at round i and let X_e be the final edge weight of e. Let $p_e = \min\left\{253\lambda_e^{-1}\epsilon^{-2}\log^2 n, 1\right\}$ where λ_e is the edge-connectivity of e and let $p'_e = \min\{4p_e, 1\}$. Let B_e be the event that the edge weight of e is not frozen until round $\lfloor \log 1/p'_e \rfloor$ and let $B_C = \cup_{e \in C} B_e$ for a set C of edges.

In the above process, freezing an edge weight at round i is equivalent to sampling an edge with probability $1/2^{i-1}$. We will use Azuma's inequality, which is an exponentially decaying tail inequality for dependent random process, instead of Lemma 5.2.2.

Lemma 5.2.4 (Azuma's inequality)**.** *A sequence of random variables $X_1, X_2, X_3, \ldots$ is called* a martingale *if for all $i \geq 1$,*

$$\mathbb{E}\left[X_{i+1} | X_i\right] = X_i.$$

If $|X_{i+1} - X_i| \leq c_i$ *almost surely for all* i*, then*

$$\mathbb{P}\left[|X_n - X_1| \geq t\right] < 2\exp\left(\frac{-t^2}{2\sum_i c_i^2}\right).$$

We prove the following lemma which is identical to Theorem 5.2.2 if no bad event B_e occurs.

Lemma 5.2.5. *Let* C *be a set of edges. For any* $p \leq p_e$ *for all* $e \in C$ *and any* $N \geq |C|$*, we have*

$$\mathbb{P}\left[\neg B_C \text{ and } \left|\sum_{e\in C} X_e - |C|\right| \geq \epsilon N\right] < 2\exp(-0.38\epsilon^2 pN) \ .$$

Proof. Suppose that we sample edges one by one and let $Y_{i,j}$ be the total weight of edges in C after j steps at round i. If $Y_{i,0} \geq |C| + \epsilon N$ for any i, we stop the sampling process.

For each step in round i, we change the edge weight from 2^{i-1} to either 2^i or 0 with equal probability. The expectation of the edge weight is 2^{i-1} and therefore, $\mathbb{E}\left[Y_{i,j}|Y_{i,j-1}\right] = Y_{i,j-1}$. In addition, there are at most $\frac{|C|+\epsilon N}{2^{i-1}}$ random variables $Y_{i,j}$ at round i since otherwise, $Y_{i,0}$ has to be greater than $|C| + \epsilon N$ and we would have stopped the sampling process. So

$$\begin{aligned}\sum_{i'<i}\sum_j |Y_{i',j} - Y_{i',j-1}|^2 &\leq \sum_{i'<i} \frac{|C| + \epsilon N}{2^{i'-1}} 2^{2(i'-1)} \\ &= \sum_{i'<i} 2^{i'-1}(|C| + \epsilon N) \leq 2^{i+1}N \ .\end{aligned}$$

Now the following inequality follows from Azuma's inequality.

$$\mathbb{P}\left[|Y_{i,0} - |C|| \geq \epsilon N\right] < 2\exp\left(-\frac{\epsilon^2 N}{2^{i+2}}\right)$$

Let $i = \lfloor \log \max\{1/(4p), 1\} \rfloor$. If B_C does not occur, $Y_{i,0} = \sum_{e \in C} X_e$. From the definition of i, $i = 0$ or $2^{-(i+2)} \geq 0.38p$. If $i = 0$, obviously $Y_{i,0} = |C|$. If $2^{-(i+2)} \geq 0.38p$, we get the desired result: $\mathbb{P}\left[|Y_{i,0} - |C|| \geq \epsilon N\right] < 2\exp(-0.38\epsilon^2 pN)$. □

Theorem 5.2.6. *Assuming access to fully independent random hash functions, there exists a single-pass, $O(\epsilon^{-2} n \log^5 n)$-space $(1 \pm \epsilon)$-sparsification algorithm in the semi-streaming model.*

Proof. By replacing Theorem 5.2.2 by Lemma 5.2.5, we can conclude that Simple-Sparsification produces a sparse graph that approximates every cut with high probability or for some edge e, B_e occurs. Consider an edge $e = (u, v)$ and some minimum u-v cut of cut value λ_e. For $i = \lfloor \log 1/p'_e \rfloor$, the expected number of edges in this cut is smaller than $k/2$ (assuming that we use a sufficiently large constant to decide k). By the Chernoff bound, e is not k-connected in G_i with high probability. By union bound, B_e do not occur for all e with high probability and we obtain the desired result. □

5.3 A Better Sparsification

In this section we present a more efficient implementation of Simple-Sparsification. See Algorithm 14. The idea is to first construct a less accurate "rough" sparsifier that we can use to estimate the connectivity of an edge. Then, rather than constructing all the H_i graphs via k-Connectivity, we can use the more efficient sparse-recovery

Algorithm 14 The SPARSIFICATION Algorithm. Steps 1-6 are performed in a single pass. Steps 7-13 are performed in post-processing.

1: Using SIMPLE-SPARSIFICATION, construct a $(1 \pm 1/2)$-sparsification H.

2: For $i \in \{1, \ldots, 2 \log n\}$, let $h_i : E \to \{0, 1\}$ be a uniform hash function.

3: **for** $i = 0$ **to** $\lfloor 2 \log n \rfloor$ **in parallel do**

4: Let G_i be the subgraph of G containing edges e such that $\prod_{j \leq i} h_j(e) = 1$.

5: For each $u \in V$, compute k-RECOVERY $(\mathbf{x}^{u,i})$ for $k = O(\epsilon^{-2} \log^2 n)$ where $\mathbf{x}^{u,i} \in \{-1, 0, 1\}^{\binom{V}{2}}$ with entries

$$\mathbf{x}^{u,i}[v, w] = \begin{cases} 1 & \text{if } u = v \text{ and } (v, w) \in G_i \\ -1 & \text{if } u = w \text{ and } (v, w) \in G_i \\ 0 & \text{otherwise} \end{cases} . \tag{5.2.1}$$

6: **end for**

7: Let $T = (V, E_T, w)$ be the Gomory-Hu tree of H.

8: **for** each edge $e \in E_T$ **do**

9: Let C be the cut induced by e and let $w(e)$ be the weight of the cut.

10: Let $j = \lfloor \log(\max\{w(e)\epsilon^2 / \log n, 1\}) \rfloor$.

11: k-RECOVERY $(\sum_{u \in A} \mathbf{x}^{u,j})$ returns all the edges in G_j that cross C with high probability.

12: Let $e = (u, v)$ be a returned edge and f be the minimum weight edge in the u-v path in the Gomory-Hu tree. If f induces C, include e to the graph sparsification with edge weight 2^j.

13: **end for**

algorithm k-Recovery in combination with the Gomory-Hu data structure.

1. *Rough-Sparsification:* We construct a $(1 \pm 1/2)$-sparsification using the algorithm in the previous section. The goal is to compute the sampling probability of edges upto a constant factor.

2. *Final-Sparsification:* For each edge $e = (u, v)$, we find a $O(1)$-approximate minimum u-v cut C_e using the rough sparsification. Based on the cut value of C_e, we compute a sampling probability p_e of e. Let $i_e = \lfloor \log 1/p_e \rfloor$. We find all edges in G_{i_e} that cross C_e. If $e \in G_{i_e}$, assign weight 2^{i_e} to e and otherwise, assign weight 0 to e.

It is important to note that dividing the process into two steps is conceptual and that both steps are performed in a single pass over the stream.

We next discuss finding the cut C_e for each e. Note that the collection of C_e has to be efficiently computable and stored in a small space. Fortunately, Gomory-Hu tree [55] is such a data structure, and it can be computed efficiently [86].

Definition 5.3.1. A tree T is a *Gomory-Hu tree* of graph G if for every pair of vertices u and v in G, the minimum edge weight along the u-v path in T is equal to the cut value of the minimum u-v cut.

Each edge in the Gomory-Hu tree induces a cut. It is a well-known fact that the cut value of such a cut is equal to the weight of the corresponding edge.

The method for finding the edges across a cut (line 11) is based an ideas developed in Chapter 3 for the spanning forest problem. Recall that the definition of $\mathbf{x}^{u,i}$ in Eq. 5.2.1 ensures that for any cut $(A, V \setminus A)$,

$$\text{support}(\sum_{u \in A} \mathbf{x}^{u,i}) = E_{G_i}(A) \ ,$$

where $E_{G_i}(A)$ is the set of edges in G_i that cross the cut. Because k-Recovery is a linear sketch, to find $E_{G_i}(A)$ (on the assumption there are at most k edges crossing the cuts) it suffices to have computed k-Recovery $(\mathbf{x}^{u,i})$ because

$$\textstyle\sum_{u \in A} k\text{-Recovery } (\mathbf{x}^{u,i}) = k\text{-Recovery } (\sum_{u \in A} \mathbf{x}^{u,i}) \ \ .$$

Theorem 5.3.2. *Assuming access to fully independent random hash functions, there exists a single-pass, $O(n(\log^5 n + \epsilon^{-2} \log^4 n))$-space $(1 \pm \epsilon)$-sparsification algorithm in the semi-streaming model. The requirement of fully independent random hash functions will be eliminated in Theorem 5.4.3.*

Proof. The algorithm can be implemented in one pass. The sparse-recovery sketches do not require any knowledge of the Gomory-Hu tree and thus can be constructed in parallel with the rough sparsification. The rest of the algorithm is performed in post-processing.

The space required to construct a $(1 \pm 1/2)$-sparsification is $O(n \log^5 n)$. The space required for each sampler is $O(k \log n)$ which is $O(\epsilon^{-2} \log^3 n)$. Since there are n such samplers per G_i, the total space required for the samplers is $O(\epsilon^{-2} n \log^4 n)$. We obtain the desired space bound by summing up both terms. □

5.4 Eliminating a Fully Independent Hash Function

Algorithm MinCut, Simple-Sparsification, and Sparsificationall relies on a uniform random hash function. However, remembering such a random hash function takes too much space. In this case, we $\tilde{\Omega}(m)$ bits. In the insertion-only, we can simply generate all the necessary random bits required for an edge and forget them after processing the edge. However, the approach of forgetting random bits associated with an edge does not work for the dynamic model since we have to reuse the same random bits for edge insertion and deletion.

In this section, we prove that we can replace the uniform random hash function with Nisan's pseudorandom generator [81]. This can be viewed as a limited independence style analysis, however this construction yields the basic result cleanly. Nisan's pseudorandom generator has the following property.

Theorem 5.4.1 (Nisan [81])**.** *Any randomized algorithm that runs in S space and using one way access to R random bits may be converted to an algorithm that uses $O(S \log R)$ random bits and runs in $O(S \log R)$ space using a pseudorandom generator.*

A pseudorandom generator is different from a hash function that only one-way read is allowed. If a random bit has been read, it cannot be read again. So Theorem 5.4.1 does not apply to the graph sparsification algorithm as it is. Instead, we rearrange

the input data so that the algorithm read each random bit only once. The argument was used first in Indyk [61].

Assume that the data stream is sorted, i.e., insertion and deletion operations of the same edge appear consecutively. For each edge, we generate necessary random bits (which are $O(\text{polylog}\, n)$ in number) and remember them until all the operations on the edge are read. In this way, we read each random bit only once and the algorithm still runs in $S = \tilde{O}(n)$ space and R is at most polynomial in n. We apply Theorem 5.4.1 to the algorithm with the sorted input stream. The graph sparsification algorithm (with the pseudorandom generator) succeeds with high probability.

Now note that because the algorithm is sketch-based, the algorithm's behavior does not change even if we change the order of the data stream. Therefore, the algorithm succeeds with high probability. The same argument also applies to the minimum cut algorithm. We have the following theorems. We also state the results for the insertion-only model here for the sake of comparison.

Theorem 5.4.2 (Variant of Theorem 5.1.1)**.** *There exists a single-pass, $O(\epsilon^{-2} n \log^5 n)$-space algorithm that $(1 \pm \epsilon)$-approximates the minimum cut in the dynamic graph stream model. The same algorithm uses $O(\epsilon^{-2} n \log^4 n)$ space in the insertion-only graph stream model.*

Theorem 5.4.3 (Variant of Theorem 5.3.2)**.** *There exists a single-pass, $O(n(\log^6 n + \epsilon^{-2} \log^5 n))$-space $(1 \pm \epsilon)$-sparsification algorithm in the dynamic graph stream model. The same algorithm uses $O(n(\log^5 n + \epsilon^{-2} \log^4 n))$ space in the insertion-only graph*

stream model.

5.5 Sparsifying a Weighted Graph

Lemma 5.5.1. *Let C be a set of edges such that edge weights are in $[1, L]$. For any $p \leq p_e$ for all $e \in C$ and any $N \geq |C|$, we have*

$$\mathbb{P}\left[\neg B_C \text{ and } \left|\sum_{e \in C} X_e - \sum_{e \in C} w_e\right| \geq \epsilon N L\right] < 2\exp(-0.38\epsilon^2 pN)$$

Lemma 5.5.1 is a variant of Lemma 5.2.5 where we have a weighted graph with edge weights in $[1, L]$ rather than an unweighted graph. The proof of Lemma 5.5.1 is identical to Lemma 5.2.5. Lemma 5.5.1 implies that by increasing sampling probability of edges by factor L (or equivalently, increasing k by factor L), we have a sparsification algorithm for a weighted graph with edge weights in $[1, L]$. This increases the space requirement and the number of edges in the graph sparsification.

Lemma 5.5.2. *There is a semi-streaming sparsification algorithm that runs in a single pass, $O(nL(\log^6 n + \epsilon^{-2}\log^5 n))$ space, and polynomial time in the dynamic graph stream model where edge weights are in $[1, L]$.*

For graphs with polynomial edge weights, we will partition the input graph into $O(\log n)$ subgraphs where edge weights are in range $[1, 2), [2, 4), \ldots$. We construct a graph sparsification for each subgraph and merge the graph sparsifications. The merged graph is a graph sparsification for the input graph. Summarizing, we have the following theorem:

Theorem 5.5.3. *There is a semi-streaming sparsification algorithm that runs in a single pass, $O(n(\log^7 n + \epsilon^{-2} \log^6 n))$ space, and polynomial time in the dynamic graph stream model where edge weights are $O(\text{poly}\, n)$.*

Remarks

One caveat of algorithms in this chapter is its reliance on Nisan's pseudorandom generator in the dynamic graph stream model. Although Nisan's pseudorandom generator allows us a relatively easy analysis of the algorithm, pseudorandom generator near-linear time rather than polylogarithmic time to generate one bit when the access pattern is not sequential [81]. Goel et al. later proved that $O(\text{poly}(\log n, \epsilon^{-1}))$-wise independence for sampling edges incident on any given vertex is sufficient to guarantee the accuracy of the sparsification and thus, improved the the running time of the algorithm to near-linear time in the dynamic graph streams [54].

Chapter 6

Primal-Dual Frameworks and Bipartite Matching

Chapter Outline: In this chapter, we present three general approaches for utilizing the primal-dual algorithms. The primal-dual algorithms maintain a primal candidate and iteratively improve the primal candidate. An oracle finds a "good direction" to improve the primal candidate. When the oracle cannot find such a direction, it provides a certificate of infeasibility. Two approaches differ in how to provide such a certificate. The first approach, "explicit verification", produces a dual solution whose objective value is $(1+O(\delta))\alpha$ which directly proves that α is (almost) infeasible. On the other hand, the other approach, "implicit verification", proves non-existence of such a direction using a constant (or polylogarithmic) factor approximation algorithm. The third approach, "dual-primal approach", concerns of the

choice of the primal LP to solve. More specifically, the approach start with the dual of the original problem and thus, has the name dual-primal. These approaches have been presented implicitly in previous research results. However, those results were mainly in the context of speed rather than space requirement which is of importance in the semi-streaming model.

We use the maximum matching problem in the bipartite graphs as an example. We also introduce some improvements of the framework through a careful analysis of the method. LP3 and LP4 are the standard primal and dual LPs for the maximum cardinality matching. LP5 and LP6 are the standard primal and dual LPs for the maximum weighted matching. Note that the edges are undirected in these LP formulations.

$$\begin{array}{lll} \max & \sum_{(i,j)\in E} x_{ij} & \\ \text{s.t} & \sum_{j:(i,j)\in E} x_{ij} \leq 1 & \forall i \in V \\ & x_{ij} \geq 0 & \forall (i,j) \in E \end{array} \quad \text{(LP3)}$$

$$\begin{array}{lll} \min & \sum_i y_i & \\ \text{s.t} & y_i + y_j \geq 1 & \forall (i,j) \in E \\ & y_i \geq 0 & \forall i \in V \end{array} \quad \text{(LP4)}$$

$$\begin{array}{lll} \max & \sum_{(i,j)\in E} w_{ij} x_{ij} & \\ \text{s.t} & \sum_{j:(i,j)\in E} x_{ij} \leq 1 & \forall i \in V \\ & x_{ij} \geq 0 & \forall (i,j) \in E \end{array} \quad \text{(LP5)}$$

$$\begin{array}{lll} \min & \sum_i y_i & \\ \text{s.t} & y_i + y_j \geq w_{ij} & \forall (i,j) \in E \\ & y_i \geq 0 & \forall i \in V \end{array} \quad \text{(LP6)}$$

The integrality gap of LP3 (and LP5) is one, since we have a bipartite graph [86]. We first present an algorithm for MCM and then generalize the algorithm for MWM.

Throughout this thesis, we round the fractional solution using an $(1-\epsilon)$ approx-

imation algorithm for MWM [34]. If we find a fractional solution $\{x_{ij}\}$ in which the number of non-zero edges is small, we collect edges $\{(i,j)|x_{ij} \neq 0\}$ and apply the algorithm which returns an integral matching of weight at least $(1-O(\epsilon))\sum_{i,j} w_{ij}x_{ij}$.

At the end of this chapter, we discuss some possible design choices in order to achieve a space- and time-efficient algorithm using primal-dual frameworks. For the simplicity of analysis, we use the insertion-only model in this chapter.

6.1 Approach I : Explicit Verification

In this section, we introduce the explicit verification approach to oracle design. We use the context of the multiplicative weights update method (see Section 2.3.1) but the approach itself is applicable to the fractional packing/covering framework as well.

Algorithm 15 is a general way to construct an oracle.[1] In Section 2.3.1, we defined the "admissibility" of a primal witness. The correctness, i.e., the convergence of the algorithm is guaranteed by two conditions of admissibility which are conflicting each other:

1. **Objective Value:** We want a witness whose objective value ($\mathbf{c}^{\mathrm{T}}\mathbf{x}$) is at least α. Assuming that the coefficients are positive, this condition prevents the oracle from assigning small values (or even zeros) to all variables.

2. **Satisfying Constraints in Expectation:** We want to satisfy the constraints

[1] This approach was demonstrated in [12] but the generalization was not made.

Algorithm 15 General Oracle Construction (Explicit). Steps 2 and 8 must be carefully designed to obtain a small width parameter.

1: Normalize $\mathbf{u}$ (normalized vector being $\mathbf{y}$) so that the objective value of dual program ($\mathbf{b}^{\mathrm{T}}\mathbf{y}$) to be α.

2: Find a subset of violated constraints in the primal program.

3: Let $S = \{i_1, i_2, \cdots, i_k\}$ be the violated constraints ($\mathbf{A}_j^{\mathrm{T}}\mathbf{y} < c_i$ for $i \in S$) and let $\Delta = \sum_{i \in S} c_i$.

4: **if** Δ is large enough **then**

5: Let $x_i = \alpha/\Delta$ for $i \in S$ and $x_i = 0$ otherwise.

6: **return** $\{x_i\}$ as a primal witness.

7: **else**

8: Construct a new $\mathbf{y}'$ from $\mathbf{y}$ so that $\mathbf{y}'$ satisfies all the constraints and $\mathbf{c}^{\mathrm{T}}\mathbf{y}' = (1 + O(\delta))\alpha$.

9: **end if**

in expectation, i.e., $\mathbf{y}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) \leq \delta \sum_j \mathbf{y}_j$. Assuming that the coefficients are positive, this condition force the oracle from assigning large values to all variables to satisfy the objective value condition.

Therefore, the problem reduces into finding x_i such that c_i is large and $\mathbf{A}_i^T\mathbf{y}$ is small. Algorithm 15 achieves that by finding violated dual constraints ($\mathbf{A}_i^T\mathbf{y} < c_i$). Lemma 6.1.1 formally shows the admissibility of the primal witness returned by Algorithm 15

Lemma 6.1.1. *If $b_i > 0$ for all constraints, Algorithm 15 returns an admissible primal witness.*

Proof. It is obvious that $\mathbf{c}^{\mathrm{T}}\mathbf{x} = \alpha$. Since the width parameter is not specified, the only remaining condition of the admissibility is $\mathbf{M}(\mathcal{D}, \mathbf{x}) \leq \delta$. We prove that $\mathbf{M}(\mathcal{D}, \mathbf{x}) \leq 0$.

$$\begin{aligned}
\mathbf{M}(\mathcal{D}, \mathbf{x}) &= \frac{1}{\sum_j y_j} \sum_j y_j \mathbf{M}(j, \mathbf{x}) = \frac{1}{\sum_j y_j} \sum_j y_j (\mathbf{A}_j \mathbf{x} - b_j) \\
&= \frac{1}{\sum_j y_j} \left(\sum_j y_j \mathbf{A}_j \mathbf{x} - \sum_j y_j b_j \right) \\
&= \frac{1}{\sum_j y_j} \left(\sum_i x_i (\mathbf{A}_i^T \mathbf{y}) - \sum_j y_j b_j \right) \\
&\leq \frac{1}{\sum_j y_j} \left(\sum_i x_i c_i - \sum_j y_j b_j \right) = \frac{1}{\sum_j y_j} (\mathbf{c}^{\mathrm{T}}\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{y}) = 0
\end{aligned}$$

The forth line comes from the fact that we assign a positive value to x_i only if the corresponding dual constraint is violated, i.e., $\mathbf{A}_i^{\mathrm{T}}\mathbf{y} < c_i$. Also we assign values to $\mathbf{x}$ so that $\mathbf{c}^{\mathrm{T}}\mathbf{x} = \alpha$ and normalize $\mathbf{y}$ so that $\mathbf{b}^{\mathrm{T}}\mathbf{y} = \alpha$. Therefore, $\mathbf{M}(\mathcal{D}, \mathbf{x}) \leq 0$. □

The admissibility of primal witnesses does not guarantee the efficiency of the overall algorithm. The efficiency rather depends on producing admissible witnesses with *a small width parameter* as well as the efficiency of the oracle itself. Larger the width parameter, more iterations are required to obtain a feasible solution for the LP. In Algorithm 15, this part still depends on problem-specific design of the oracle. However, it provides an intuition how to achieve a small width parameter. Suppose that there are many violated constraints, i.e., we have large value of Δ. Then, Algorithm 15 will assign relatively small values to x_i's which implies small values of

$\mathbf{M}(i, \mathbf{x})$. On the other hand, if we have a small number of violated constraints, it would be possible to alter $\mathbf{y}$ in Algorithm 15 in small amount and obtain a feasible dual solution $\mathbf{y}'$. In Section 6.1.1 through Section 6.1.3, we demonstrate this approach and some improvements.

6.1.1 Warming Up: $O(\frac{1}{\varepsilon^3}\log n)$-pass Algorithms

In this section, we provide a $(1-\varepsilon)$-approximation algorithm for bipartite MCM and MWM that uses $O(\frac{1}{\varepsilon^3}\log n)$ passes. We will use the multiplicative weights update method reviewed in Section 2.3.1. Recall that the method provides a solution to the primal problem, provided that the oracle does not fail.

The Simple Case of MCM

We apply the multiplicative weights update method with the oracle provided in Algorithm 16. Recall that if the oracle does not fail, Algorithm 3 returns a feasible solution for LP3 after T iterations.

Recall that we can compute a maximal matching in one pass, $O(m)$ time and $O(n)$ space in the insertion-only model. It is trivial to observe that any maximal matching is a 2 approximation to the maximum cardinality matching.

Lemma 6.1.2. *If $\Delta \geq \delta\alpha$, the oracle described in Algorithm 16 returns an admissible solution with $\ell = 1$ and $\rho = 1/\delta$.*

Proof. Note $x_{ij} = \alpha/|S|$ for $(i,j) \in S$ and $x_{ij} = 0$ for $(i,j) \notin S$. Therefore, it is

Algorithm 16 Oracle for LP4. The input is $\{u_i^t\}_{i \in V}, \alpha$.

Let $y_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$. Let $E_{violated} = \{(i,j) | y_i + y_j < 1\}$.

Find a **maximal matching** S in $E_{violated}$.

Let $\Delta = |S|$.

if $\Delta < \delta\alpha$ **then**

  For each $(i,j) \in S$, increase y_i and y_j by 1. Observe that $\mathbf{y}$ is feasible for LP4 and $\sum_i y_i \leq (1+2\delta)\alpha$.

  Return $\mathbf{y}$ and report failure.

else

  Return $x_{ij} = \alpha/\Delta$ for $(i,j) \in S$ and $x_{ij} = 0$ otherwise.

end if

obvious that $\sum_{(i,j)\in E} x_{ij} = \alpha$. Since the vector $\mathbf{c}$ is all 1, we have $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \alpha$.

For each edge $(i,j) \in S$ we have $y_i + y_j < 1$, and therefore we have $\sum_{(i,j)\in S} x_{ij}(y_i + y_j) < \alpha$. Therefore $\sum_{(i,j)\in E} x_{ij}(y_i + y_j) = \sum_{(i,j)\in S} x_{ij}(y_i + y_j) < \alpha$. This rewrites to $\sum_i y_i \sum_{j:(i,j)\in E} x_{ij} < \alpha$. Observe that $\sum_i y_i = \alpha$ and therefore $\sum_i y_i(\sum_{j:(i,j)\in E} x_{ij} - 1) < 0$.

Thus $\mathbf{M}(\mathcal{D}^t, \mathbf{x}) = \frac{1}{\alpha}\sum_i y_i \mathbf{M}(i, \mathbf{x}) \leq 0 < \delta$. Now $\mathbf{M}(i, \mathbf{x}) = \sum_{j:(i,j)\in E} x_{ij} - 1 \geq -1$. Since S is a matching, for every i at most one $x_{ij} \neq 0$ and moreover $x_{ij} \leq 1/\delta$ (otherwise the oracle has failed). Therefore $-1 \leq \mathbf{M}(i, \mathbf{x}) \leq 1/\delta$ and the solution is admissible. □

Lemma 6.1.3. *If $\Delta < \delta\alpha$, Algorithm 16 returns a feasible solution for LP4 with*

value at most $(1+2\delta)\alpha$.

Proof. Consider $(i,j) \in E$ such that $y_i + y_j < 1$. Since S was maximal, there exists an edge in S that is incident on either i or j. So y_i or y_j is increased by at least 1 and the constraint corresponding to edge (i,j) is satisfied. For each edge $(i',j') \in S$, we increase the objective value by 2 and $|S| \leq \delta\alpha$. Since we started with $\sum_i y_i = \alpha$, the solution returned has value at most $(1+2\delta)\alpha$ after the increase. □

Theorem 6.1.4. *For any* $\varepsilon \leq \frac{1}{2}$ *let* $T = O(\frac{1}{\varepsilon^3}\log n)$. *Using* $T+1$ *passes and space* $O(\frac{nT}{\varepsilon})$ *and time* $O(\frac{mT}{\varepsilon})$ *time we can find a* $(1-\varepsilon)$ *approximation to the maximum cardinality matching in bipartite graphs. This implies a* $\frac{2}{3}(1-\varepsilon)$ *result for general graphs using the integrality gap results of [47, 48].*

Proof. We use the first pass to compute OPT to within factor 2 — this follows from the fact that any maximal matching is a 2 approximation to the maximum cardinality matching. Suppose the size of the maximal matching we found is q. We try all possible values of $\alpha = (1+\frac{\varepsilon}{3})^j q$ where $j \geq 0$ and $\alpha \leq 2q(1+\frac{\varepsilon}{3})$ in parallel. This corresponds to $O(\frac{1}{\varepsilon})$ guesses of α.

Let $\delta = \varepsilon/12$. Therefore $\epsilon = \min\{\frac{\delta}{4}, \frac{1}{2}\} = \varepsilon/48$ since $\ell = 1$. Note, the parameters ϵ, ε are different. We now apply the Algorithm 3 using Algorithm 16 as the oracle.

Let α_0 to be the smallest value of α which is above OPT, i.e., $\alpha_0 \geq OPT > \alpha_0/(1+\frac{\varepsilon}{3})$. Consider $\alpha \leq \alpha_0/(1+\frac{\varepsilon}{3})^2 < OPT/(1+\frac{\varepsilon}{3})$. For any such value of α it is impossible that the oracle fails since we return a feasible primal solution of value at most $(1+2\delta)\alpha = (1+\varepsilon/6)\alpha \leq (1+\frac{\varepsilon}{6})OPT/(1+\frac{\varepsilon}{3}) < OPT$. Therefore if we

consider the largest value of α for which we **do not** return a feasible primal solution, that value must satisfy $\alpha \geq \alpha_0/(1+\frac{\varepsilon}{3})^2$. Let this value be α^*. Using Corollary 2.3.3, after T iterations we have a feasible primal solution $\tilde{\mathbf{x}}$. Note all $b_i = 1$ and $4\delta = \varepsilon/3$. By construction $\sum_{(i,j)\in E} x_{ij}^t = \alpha^*$ for every t. Therefore,

$$\sum_{(i,j)\in E} \tilde{x}_{ij} \geq \frac{1}{1+4\delta}\alpha^* = \frac{\alpha^*}{1+\frac{\varepsilon}{3}} \geq \frac{\alpha_0}{(1+\frac{\varepsilon}{3})^3} \geq (1-\varepsilon)\alpha_0 \geq (1-\varepsilon)OPT$$

The time and space bounds follow easily. To find the actual matching: Let $\varepsilon = \varepsilon'/2$ and we run the above steps to find a fractional solution. We find T matchings before we return the best fractional solution, there are at most $m' = O(nT)$ non-zero entries in the solution. Focus on the graph G' defined by these edges only. The fractional solution of the original graph remains a fractional solution in G'. We now have random access to these edges in G' and can find a $(1-\varepsilon)$ approximation to the best matching contained in these edges (which is at least the same value as the fractional solution, we use the integrality of the bipartite matching polytope) in time $\tilde{O}(m')$ using known algorithms [59, 76, 65]. The overall approximation is $(1-\varepsilon)^2 \geq (1-\varepsilon')$. □

The oracle, Algorithm 16 is an example of the explicit approach (Algorithm 15). The intuition behind the oracle will be used in the subsequent algorithms for bipartite MCM and MWM. In Algorithm 16 we must choose a subset S of edges which balances two critical properties:

Admissibility : Each vertex i is adjacent to at most one edge in S. The weights assigned to the edges in S (note, they are identical for a specific iteration t) define the parameters ℓ, ρ. These parameters determine the number of iterations.

Verification : Focusing on the violations in the primal solution allows us to produce a feasible primal solution and verify α. For each violated edge (i, j) in the primal solution, we pick at least one adjacent edge.

Any **maximal** matching in $E_{violated}$ satisfies both conditions. Since we consider the violated edges only, the algorithm is natural. Observe that the framework operates on primal violations whereas the oracle operated on dual violations. In a sense, the problem of finding the maximum matching problem in bipartite graphs reduces to the problem of repeatedly finding maximal matchings in subgraphs defined by dual violations (corresponding to the edges). These violations can be easily defined by a simple filtering conditions, for example, does the input edge satisfy $y_i + y_j < 1$ using the current solution $\mathbf{y}$ of the dual, and can be implemented in the semi-streaming model. We now proceed to discuss weighted graphs — observe that weights will also arise naturally in the unweighted case as we improve Theorem 6.1.4.

The Not So Simple Case of MWM

Note that the input for weighted problems in the semi-streaming model is a sequence of tuples $\{(i, j, w_{ij})\}$ and the weights do not have to be stored. We can easily compute a maximal matching in a single pass using $O(n)$ space and $O(m)$ time. It is shown in [43] that we can compute a 1/6 approximation to the maximum weighted matching in a single pass using $O(n)$ space and $O(m)$ time.

In a weighted graph, the verification condition must be strengthened to handle

the complications introduced by edge weights. In LP3, if we increase y_i by 1, then all the edges adjacent to i are satisfied. Therefore if only a few primal constraints were violated then we could produce a primal feasible solution which is close to α. It is not true in LP5, we now have to increase y_i by the amount of violation. However trying to fix the verification condition by itself does not help, any change also has to ensure the admissibility condition and a larger increase in y_i corresponds to larger ρ.

Let $w(S) = \sum_{(i,j)\in S} w_{ij}$ denote the total weight for any set of edges S. The oracle will search for a set S of edges which satisfy the following:

Weighted Admissibility of S : There exists a matching S' contained in S such that $w(S') = \Omega(w(S))$. We use S' to construct a dual witness. Since S' is a matching we will have some control over ℓ, ρ.

Weighted Verification of S : For each violated edge (i, j), we pick at least one edge adjacent to it whose weight is $\Omega(w_{ij})$. We need all of S to produce an upper bound of the primal solution.

In order to satisfy the modified conditions, we partition the edges depending on their weights.

Definition 6.1.5. An edge (i, j) is in **tier** k if $\alpha/2^k < w_{ij} \leq \alpha/2^{k-1}$.

Algorithm 17 is the oracle for LP6. Before we prove the admissibility and verification conditions we prove a useful lemma which is a property of the constraints.

Algorithm 17 Oracle for LP6.

Let $y_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$.

Let $E_{violated,k} = \{(i,j) | y_i + y_j < w_{ij}, \alpha/2^k < w_{ij} \leq \alpha/2^{k-1}\}$.

Find a maximal matching S_k in $E_{violated,k}$ for each $k = 1, \cdots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.

Let $S = \cup_k S_k, \Delta = w(S)$.

if $\Delta < \delta\alpha$ **then**

  For each $(i,j) \in S$, increase y_i and y_j by $2w_{ij}$.

  Further increase every y_i by $\frac{\delta\alpha}{n}$.

  Return $\mathbf{y}$ and report failure.

else

  $S' \leftarrow \emptyset$.

  repeat

    Pick a heaviest edge (i,j) from S and add it to S'

    Eliminate all edges adjacent to i or j from S.

  until $S = \emptyset$

  Return $x_{ij} = \alpha/w(S')$ for $(i,j) \in S'$ and $x_{ij} = 0$ otherwise.

end if

Lemma 6.1.6. *If* $\mathbf{y}$ *satisfies* $\sum_{(i,j)\in E} w_{ij}y_{ij} = \alpha$ *then for any weights* $\mathbf{u}^t$*, if we have* $x_i^t = \alpha u_i^t/(\sum_j u_j^t)$ *then* $\mathbf{M}(\mathcal{D}^t, \mathbf{y}) = \frac{1}{\alpha}\left(\sum_{(i,j)\in E} y_{ij}(x_i^t + x_j^t - w_{ij})\right)$.

Proof. From the definition of $\mathbf{M}(\mathcal{D}^t, \mathbf{y})$ we have:

$$\begin{aligned}\alpha\mathbf{M}(\mathcal{D}^t, \mathbf{y}) &= \sum_i x_i^t(\sum_{j:(i,j)\in E} y_{ij} - 1) = \sum_{(i,j)\in E} y_{ij}(x_i^t + x_j^t) - \sum_i x_i^t \\ &= \sum_{(i,j)\in E} y_{ij}(x_i^t + x_j^t) - \sum_{(i,j)\in E} w_{ij}y_{ij} = \sum_{(i,j)\in E} y_{ij}(x_i^t + x_j^t - w_{ij})\end{aligned}$$

The lemma follows. □

Lemma 6.1.7. *(Weighted Admissibility.) The matching* $S' \subseteq S$ *constructed by Algorithm 17 satisfies* $w(S') \geq w(S)/5$*. As a consequence, if* $\Delta \geq \delta\alpha$ *the Algorithm 17 returns an admissible dual witness with* $\rho = \frac{5}{\delta}$ *and* $\ell = 1$*.*

Proof. Observe that in S there is at most one edge incident to i from each tier of weight. Consider the matching S' constructed by Algorithm 17. Suppose that $(i,j) \in S'$ is in tier k and consider the edges $A_i = \{(i,j')|(i,j') \in S, j \neq j'\}$ which are eliminated from S by the inclusion of this edge (i,j) in S'. Each element in A_i has a lower weight than (i,j); otherwise we would have chosen that eliminated edge instead of (i,j) (the weights cannot be equal since there cannot be any other edge from tier k which is incident on i). Therefore the edges in A_i belong to tiers numbered $k+1$ or larger (since they have a lower weight). The weight of any ignored edge in tier $k+q$ can be upper bounded by $w_{ij}/2^{q-1}$. These weights add up to $2w_{ij}$ since we have at most one edge from each tier. Therefore the sum of the weights of the edges in A_i, A_j amount to at most $4w_{ij}$. Summing over all $(i,j) \in S'$, $w(S - S') \leq 4w(S')$ and thus $w(S') \geq w(S)/5$.

For the second part of the lemma, observe that $x_{ij} = \alpha/w(S') \leq 5\alpha/w(S) = 5\alpha/\Delta \leq \frac{5}{\delta}$. The parameter ℓ remains 1 and ρ is $\frac{5}{\delta}$ due to the same reason as in the proof of Lemma 6.1.2. We observe that (since $y_{ij} = 0$ for $(i,j) \notin S'$);

$$\mathbf{c}^T\mathbf{y} = \sum_{(i,j)\in E} w_{ij}y_{ij} = \sum_{(i,j)\in S'} w_{ij}y_{ij} = \sum_{(i,j)\in S'} w_{ij}\frac{\alpha}{w(S')} = \frac{\alpha}{w(S')}w(S') = \alpha$$

Applying Lemma 6.1.6 we immediately get $\alpha\mathbf{M}(\mathcal{D}^t, \mathbf{y}) = \sum_{(i,j)\in E} y_{ij}(x_i^t + x_j^t - w_{ij})$. Now if $x_i^t + x_j^t - w_{ij} \geq 0$ then $y_{ij} = 0$ by construction. Or in other words, $\mathbf{M}(\mathcal{D}^t, \mathbf{y}) \leq 0 \leq \delta$. The lemma follows.. □

Lemma 6.1.8. *(Weighted Verification.) For every violated edge (i,j) in one of the $O(\log n)$ tiers, we pick at least one edge in S adjacent to (i,j) whose weight is at least $w_{ij}/2$. As a consequence, if $\Delta < \delta\alpha$ then the algorithm returns a feasible primal solution for LP6 with value at most $(1+5\delta)\alpha$.*

Proof. Suppose that (i,j) is violated in tier k. Then, since S_k is a maximal matching, we must have chosen at least one edge in S_k which is adjacent to i or j. The weight of that chosen edge in S_k has to be at least $w_{ij}/2$ since the weights of two edges that belong to the same tier differ at most by a factor of 2.

For the second part of the proof we follow the argument in Lemma 6.1.3, with one change. Suppose that $y_i + y_j < w_{ij}$ for an edge $(i,j) \in E$. We have two cases, either the edge was chosen in one of the tiers (say k) or $w_{ij} \leq \delta\alpha/n$. The second case is easier, since we increase each y_i by at least $\delta\alpha/n$, we definitely satisfy the constraint for (i,j) in this case.

For the first case, observe that in the first part of the lemma we proved that we selected an edge $e \in S$ incident on i or j with weight $w_e \geq w_{ij}/2$. Therefore we increased x_i or x_j by at least $2w_e \geq w_{ij}$. We satisfy the constraint for (i, j) in this case as well. Therefore $\mathbf{x}$ is feasible.

For each edge $(i, j) \in S$, we increase $\sum_i y_i$ by $4w_{ij}$. Therefore over all the edges we increase $\sum_i y_i$ by $4w(S) = 4\Delta$. Since we started with $\sum_i y_i = \alpha$ we have $\sum_i y_i = \alpha + 4\Delta < (1+4\delta)\alpha$ after we increase x_i based on the edges. We now have an additional increase in y_i which adds $\delta\alpha$ to $\sum_i y_i$. The lemma follows. $\square$

The rest of the argument is almost identical to MCM and proof of Theorem 6.1.4 with four changes: (i) ρ increases to $\frac{5}{\delta}$ from $\frac{1}{\delta}$ (ii) we need to set $\delta = \varepsilon/30$ since the primal feasible solution returned is at most $(1 + 5\delta)\alpha$ (iii) we start with the 1/6 approximation provided by [43] which uses $O(n)$ space and (iv) for the final rounding scheme we use the recent result of [34]. The space bound increases since the oracle now uses $O(n \log n)$ space due to the $O(\log n)$ tiers (and as before we have $O(\frac{1}{\varepsilon})$ oracles being run in parallel).

Theorem 6.1.9. *For any $\varepsilon \leq \frac{1}{2}$ in $T = O(\frac{1}{\varepsilon^3} \log n)$ passes, and $O(\frac{nT}{\varepsilon} + \frac{n}{\varepsilon} \log n)$ space we can compute a $(1 - \varepsilon)$ approximation for maximum weighted matching in bipartite graphs.*

6.1.2 Reducing Space and Passes

So far we have not used the fact that we are trying to solve the same LP for different guesses of the parameter α. Moreover we have used one pass for each invocation of the oracle. The number of passes is equal to the number of iterations plus one; the first pass is used to guess the values of α.

In this section, we first reduce the space required to manage the multiple guesses of α. Subsequently, we reduce the number of passes by executing multiple iterations of the algorithm in one pass – this can be viewed as making a "step" which is significantly larger than what is provided by the basic analysis in the previous section. We focus on the weighted case.

Reducing the Space Requirement

In what follows we show how to preserve the admissibility condition across different values of the guessed parameter α, and run the $O(\frac{1}{\varepsilon})$ guesses (in Theorem 6.1.9) in parallel without increasing the space requirement by a factor $O(\frac{1}{\varepsilon})$. The key intuition is that we are trying to find feasible solutions for the same instance of LP5 but different values of the objective function. If in a single iteration we make progress for a large value of α then we also make progress for a smaller value of α.

Observe that the proofs of Theorem 6.1.4 and 6.1.9 use α^* which is the largest value of α for which we have not produced a feasible primal solution. Suppose that we can prove that we would make the same choices for different values of α. Then,

when we produce a feasible primal solution for some guess of α (the oracle fails), it may be that for a smaller guess of α the oracle does not fail. We can continue with the smaller guess of α, as if the larger guess was never made! Therefore we will avoid running separate oracles for the different guesses of α and thereby save space. We begin with the following definition:

Definition 6.1.10. A sequence $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^t$ is **admissible** if all $\mathbf{x}^s$ for $1 \leq s \leq t$ are admissible when we apply $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^t$ in the given order. Recall that admissibility is a property of both $\mathbf{u}, \mathbf{x}$ (Definition 2.3.1).

Lemma 6.1.11. *Let α, α' be guesses of the optimal solution with $\alpha > \alpha'$. If a sequence $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^t$ is admissible for α, the sequence is also admissible for α'.*

Proof. Consider running the two copies of the Algorithm 3: for the values of α and α'. Observe that $\mathbf{M}(i, \mathbf{x})$ only depends on $\mathbf{x}$ and therefore the parameters ℓ, ρ do not depend on α, α'. Moreover the actual weights of the edges do not change and therefore for any vector $\mathbf{x}$ if $\mathbf{c}^T\mathbf{x} \geq \alpha$ then $\mathbf{c}^T\mathbf{x} > \alpha'$.

Therefore to show admissibility, it suffices to prove that $\mathbf{M}(\mathcal{D}^q, \mathbf{x}^q) \leq \delta$ for all $q \leq t$ for the smaller value α' assuming that $\mathbf{x}^q$ satisfied $\mathbf{M}(\mathcal{D}^q, \mathbf{x}^q) \leq \delta$ for all $q \leq t$ for the larger value α. We prove this using induction.

Initially $\mathbf{u}$ is same for both copies of the algorithm (as described so far, we have used $u_1^i = 1$, but we will be changing this in the next section). Now $\mathbf{M}(\mathcal{D}^1, \mathbf{x}^1) = \frac{1}{\sum_j u_j^1} \sum_i u_i^1 \mathbf{M}(i, \mathbf{x}^1)$ and is independent of α. Therefore $\mathbf{x}^1$ is admissible for α'. This proves the base case.

Suppose that we have proven the hypothesis up to $q = k$ and we apply $\mathbf{x}^1, \cdots, \mathbf{x}^k$ to both the algorithms corresponding to α and α'. Observe that $\rho, \mathbf{M}(i, \mathbf{x})$ are unchanged and therefore the weights u_i^{k+1} is the same for both α and α'. But $\mathbf{M}(\mathcal{D}^{k+1}, \mathbf{x}^{k+1}) = \frac{1}{\sum_j u_j^{k+1}} \sum_i u_i^{k+1} \mathbf{M}(i, \mathbf{x}^{k+1})$ and for all i both algorithms have the same value of $\frac{1}{\sum_j u_j^{k+1}} u_i^{k+1}$ and $\mathbf{M}(i, \mathbf{x}^{k+1})$ since these quantities are independent of α. Therefore $\mathbf{x}^{k+1}$ is also admissible for α'. The lemma follows by induction. $\square$

The algorithm: We start with α being the upper bound of the maximum matching. Each time the oracle fails, we reduce α by $(1 + \frac{\varepsilon}{3})$ factor while keeping the weights of constraints and $\{\mathbf{x}^t\}$ fixed which were computed so far. This is possible since the sequence of $\mathbf{x}$ remains admissible with the same width parameter. The total number of successful iterations remains the same but we need an additional iteration for each time the oracle reports failure. However we only have to provision for solving one copy of the oracle.

We can now show that Theorem 6.1.9 holds with space $O(n(T + \log n))$, but uses $T' = T + O(\frac{1}{\varepsilon})$ passes. Formally,

Theorem 6.1.12. *For any $\varepsilon \leq \frac{1}{2}$ in $T = O(\frac{1}{\varepsilon^3} \log n)$ passes, and $O(n(T + \log n))$ space we can compute a $(1 - \varepsilon)$ approximation for maximum weighted matching in bipartite graphs.*

Algorithm 18 Improved Algorithm for MWM (reducing space).

1: In one pass, find a 6 approximate maximum matching using [43] and let α_0 be the weight of the matching.

2: $u_i^1 = 1$ for all $i \in [n]$ and $\alpha = 6\alpha_0$

3: **for** $t = 1$ **to** T **do**

4: Given u_i^t, run the oracle (Algorithm 17).

5: If the oracle failed decrease α by factor $(1 + \frac{\varepsilon}{3})$ and repeat line 4.

6: Let $\mathbf{M}(i, \mathbf{x}^t) = \mathbf{A}_i \mathbf{x}^t - b_i$. ($\mathbf{x}$ is an admissible dual witness now)

7: $$u_i^{t+1} = \begin{cases} u_i^t (1+\epsilon)^{\mathbf{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathbf{M}(i,\mathbf{x}^t) \geq 0 \\ u_i^t (1-\epsilon)^{-\mathbf{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathbf{M}(i,\mathbf{x}^t) < 0 \end{cases}$$

8: **end for**

9: Output $\frac{1}{T}\frac{1}{1+4\delta}\sum_t \mathbf{x}^t$.

Reducing the number of passes

Consider the two conditions for the oracle given in the previous section, and for the sake of example, consider the cardinality case. Suppose that we just performed an update based on a primal witness $\mathbf{x}$. Observe that $\mathbf{y}^t = \mathbf{x}(\mathbf{u}^t)$ and for the next step, the admissibility condition ($\mathbf{M}(\mathcal{D}^t, \mathbf{x}) \leq 0 \leq \delta$) remains satisfied as long as the edges (i, j) in support($\mathbf{x}$) satisfy $y_i^t + y_j^t < 1$. Therefore as a new approach, we do not invoke the oracle again as long as we have such a solution. In other words, we can use the same matching returned as a primal witness for multiple iterations or until one of its edges satisfies the corresponding dual constraint $y_i + y_j \geq 1$.

Therefore it appears that we can simulate multiple iterations in a single pass. But

if $y_i + y_j$ is close to 1 then this idea need not be useful because we may satisfy that edge in a single step. Observe that this idea automatically brings up the notion of weights even in the context of MCM. The high-level idea for the oracle is similar to the construction in Section 6.1.1 – but there are significant differences and two major issues arise.

- First, we cannot use uniform values for the entries of $\mathbf{y}$ as in Section 6.1.1, even in the setting of MCM. Suppose that S contains (i, j) and (i', j') where $1 - y_{i'} - y_{j'}$ is greater than $1 - y_i - y_j$. If we assign large values to x_{ij} and $x_{i'j'}$, it decreases the number of iterations per pass (due to normalization the y_i for the matched edges rise quickly, and we satisfy the constraint). If we assign small values to x_{ij} and $x_{i'j'}$, it increases the total number of iterations and it may also result in inadmissible $\mathbf{x}$, i.e., $\mathbf{c}^T\mathbf{x} < \alpha$.

- Second, we have to modify the verification condition in Section 6.1.1 so that the condition handles the values of $w_{ij} - y_i - y_j$ and keep the increase of the solution minimal. For example, (again using the cardinality case as an example) increasing the value of y_i and y_j less than 1 in the verification step can result in an infeasible primal solution. On the other hand, increasing y_i and y_j by 1 can result in a larger approximation factor.

In what follows, we avoid both the issues by defining the tier of an edge based on the violation instead of the edge weight. Moreover we ensure that for different edges (i, j) the x_{ij} values are different — this can be viewed as setting $w_{ij}x_{ij}$ proportional to

the violation in (i, j). Therefore the accounting for the admissibility and verification conditions are different.

Definition 6.1.13. Define $v_{ij} = w_{ij} - y_i - y_j$ to be the **(dual) violation** of an edge $(i, j) \in E$ (the edge is not violated if $v_{ij} < 0$). An edge (i, j) is in **violation-tier** k if $\alpha/2^k < v_{ij} \leq \alpha/2^{k-1}$. Observe that if $\alpha \geq \max_{i,j} w_{ij}$ then every edge belongs to a violation-tier numbered by a natural number. For any set of edges S, define $V(S) = \sum_{(i,j)\in S} v_{ij}$. Define $\tilde{v}_{ij} = \max\{v_{ij}/w_{ij}, 0\}$.

The improved oracle is given in Algorithm 19.

Lemma 6.1.14. *In Algorithm 19, if $\Delta_V \geq \delta\alpha$ then we have a matching $\mathbf{x}$ such that $\sum_{(i,j)\in E} w_{ij}x_{ij} = \alpha$, and for all i either $\mathbf{M}(i, \mathbf{x}) = -1$ or $\mathbf{M}(i, \mathbf{x}) \leq \frac{5}{\delta}\tilde{v}_{ij} - 1$ where $(i, j) \in S'$. As a consequence if $\Delta_V \geq \delta\alpha$ then Algorithm 19 returns an admissible solution with $\ell = 1$ and $\rho = \frac{5}{\delta}$.*

Proof. The proof is similar to the proof of Lemma 6.1.7, except that we will use violations (whereas the proof of Lemma 6.1.7 used the weights). Observe that since $x_{ij} = 0$ if $(i, j) \notin S'$ we have

$$\sum_{(i,j)\in E} w_{ij}x_{ij} = \sum_{(i,j)\in S'} w_{ij}x_{ij} = \sum_{(i,j)\in S'} w_{ij}\frac{\tilde{v}_{ij}\alpha}{\Delta'_V} = \frac{\alpha}{\Delta'_V}\sum_{(i,j)\in S'} v_{ij} = \alpha$$

Note $\sum_i y_i = \alpha = \sum_{(i,j)\in E} w_{ij}x_{ij}$ (observe α is not changed within an iteration).

Now suppose that $(i, j) \in S'$ is in violation-tier k and consider edges adjacent to i. All of them are in violation-tier $k + 1$ or higher and for each such tier we have at most two edges (one adjacent to i and one adjacent to j) because we pick a maximal

Algorithm 19 Improved Oracle for LP6.

Let $y_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$.

Let $E_{violated,k} = \{(i,j) | (i,j) \text{ is in violation-tier } k\}$. for $k = 1, \cdots, K = \lceil \log_2 \frac{n}{\delta} \rceil$

Find a **maximal matching** S_k in each $E_{violated,k}$.

Let $S = \cup_k S_k$and $\Delta_V = V(S)$.

if $\Delta_V < \delta\alpha$ **then**

  For each $(i,j) \in S$, increase y_i and y_j by $2v_{ij}$.

  Further increase all y_i by $\frac{\delta\alpha}{n}$. Return $\mathbf{x}$ and report failure.

else

  $S' \leftarrow \emptyset$.

  repeat

    Pick a edge (i,j) from S with largest v_{ij} and add it to S'

    Eliminate all edges adjacent to i or j from S.

  until $S = \emptyset$

  Let $\Delta'_V = V(S')$.

  Return $x_{ij} = \tilde{v}_{ij}\alpha/\Delta'_V$ for $(i,j) \in S'$ and $x_{ij} = 0$ otherwise.

end if

matching for each violation-tier. So the total violation of edges that are eliminated by (i,j) is at most $\sum_{k'=1}^{\infty} \frac{2v_{ij}}{2^{k'-1}} \leq 4v_{ij}$. This shows that $V(S - S') \leq 4V(S')$ and therefore $V(S') \geq V(S)/5$. Hence $\alpha/\Delta'_V \leq 5\alpha/\Delta_V \leq 5/\delta$ (otherwise the oracle has failed).

Now $\mathbf{M}(i,\mathbf{x}) = \sum_{j:(i,j)\in E} x_{ij} - 1$. Therefore if i is unmatched in S' we have $M(i,\mathbf{x}) = -1$. Otherwise $M(i,\mathbf{x}) = x_{ij} - 1 = \tilde{v}_{ij}\alpha/\Delta'_V - 1 \leq \frac{5}{\delta}\tilde{v}_{ij} - 1$ where $(i,j) \in S'$. This proves the first part of the lemma.

For the second part observe that $\mathbf{c}^T\mathbf{x} = \sum_{(i,j)\in E} w_{ij}x_{ij} = \frac{\alpha}{\Delta'_V}\sum_{(i,j)\in S'} w_{ij}\tilde{v}_{ij} = \alpha$. Using Lemma 6.1.6 we have $\alpha\mathbf{M}(\mathcal{D}^t,\mathbf{y}) = \sum_{(i,j)\in E} x_{ij}(y_i^t + y_j^t - w_{ij})$. Now if $y_i^t + y_j^t - w_{ij} \geq 0$ then $x_{ij} = 0$ by construction. Therefore $\alpha\mathbf{M}(D^t,\mathbf{x}) \leq 0$ and so $\mathbf{M}(i,\mathbf{x}) \leq 0 \leq \delta$. Finally $0 \leq \tilde{v}_{ij} \leq 1$ and therefore $-1 \leq \mathbf{M}(i,\mathbf{x}) \leq \frac{5}{\delta}$. The lemma follows. □

Lemma 6.1.15. *If $\Delta_V < \delta\alpha$, then Algorithm 19 returns a feasible solution for LP6 with value at most $(1+5\delta)\alpha$.*

Proof. The proof follows similar arguments as in the proof of Lemma 6.1.8; except that we use violations in this proof instead of weights (as in the proof of Lemma 6.1.8). We consider the normalized weights $\mathbf{y}$ as a primal candidate for LP6. If the oracle fails, we augment $\mathbf{y}$ to obtain a feasible primal solution with a small increase.

Suppose that (i,j) is in violation-tier k. Since S_k was maximal, there exists an edge that is adjacent to either i or j. So x_i or x_j is increased by at least $\alpha/2^{k-1}$ and the constraint is satisfied. If (i,j) did not belong to any of the violation-tiers then its

violation was less than $\delta\alpha/n$ and since x_i, x_j are increased by $\delta\alpha/n$ this constraint is also satisfied.

For each edge $(i,j) \in S$, we increase the objective value by $4v_{ij}$ and $\sum_{(i,j)\in S} v_{ij} = \Delta_V < \delta\alpha$. Finally, we increase all y_i by $\delta\alpha/n$ which increases the objective value by at most $\delta\alpha$. So our primal solution has value at most $(1+5\delta)\alpha$. □

The next lemma is the central idea in this subsection. Consider running Algorithm 18, but with Algorithm 19 as the oracle instead of Algorithm 17. Based on Lemmas 6.1.14 and 6.1.15, and Theorem 6.1.12 we know that in $T = O(\frac{1}{\varepsilon^3}\log n)$ iterations we will find a $(1-\epsilon)$ approximation of the maximum weighted matching. Surprisingly, we will now prove that even if we do not update the witness $\mathbf{x}$ for $\frac{1}{\delta}$ steps, the witness remains admissible!

Lemma 6.1.16. *Consider running Algorithm 18, but with Algorithm 19 as the oracle instead of Algorithm 17. If the dual witness* $\mathbf{x}$ *computed by the Algorithm 19) in iteration* t *is admissible, then* $\mathbf{x}$ *remains admissible for all iterations* $t+q$ *where* $q \leq 1/\delta$.

Proof. Since $\mathbf{x}$ was admissible (in any iteration) $-\ell \leq \mathbf{M}(i,\mathbf{x}) \leq \rho$. Since $\mathbf{y}$ was computed in iteration t we know from the proof of Lemma 6.1.14 that $\sum_{(i,j)\in E} w_{ij}x_{ij} = \alpha$ and $\mathbf{M}(i,\mathbf{x})/\rho \leq \tilde{v}_{ij}^t$. We use $\tilde{v}_{ij}^t$ to indicate that this is the fractional violation that was used to determine x_{ij}. Note that $w_{ij}(1-\tilde{v}_{ij}^t) = y_i^t + y_j^t$. Also note $0 < \tilde{v}_{ij}^t \leq 1$ for a violated edge.

Note that even though we are not updating $\mathbf{x}$ across the iterations, $\mathbf{u}, \mathbf{y}$ are being updated (using the same $\mathbf{x}$ at every step) and we need to prove $\mathbf{M}(\mathcal{D}^{t+q}, \mathbf{x}) \leq \delta$ for every $t+q$ where $q \leq 1/\delta$.

Then by Lemma 6.1.6,

$$\mathbf{M}_q(\mathcal{D}^{t+q}, \mathbf{x}) = \frac{1}{\alpha} \sum_{(i,j)\in E} w_{ij} x_{ij} \frac{y_i^{t+q} + y_j^{t+q} - w_{ij}}{w_{ij}}$$

and since $\sum_{(i,j)\in E} w_{ij} x_{ij} = \alpha$, it is sufficient to show that $\frac{y_i^{t+q}+y_j^{t+q}-w_{ij}}{w_{ij}} \leq \delta$ for $x_{ij} \neq 0$. This means that we need to focus on the edges in the matching S' only, since all other edges have $x_{ij} = 0$.

In the following $\mathbf{y}^{t+q}$ refers to the dual candidate after q iterations of Algorithm 18 using the witness $\mathbf{x}$ at every step from iteration t. Note $\epsilon = \delta/4\ell$ (since we will eventually set $\delta \ll 1$ based on ε, and therefore $\epsilon \ll \frac{1}{2}$) and $\ell = 1$. Note that,

$$y_i^{t+1} \leq y_i^t \frac{(1+\delta/4)^{\tilde{v}_{ij}^t}}{(1-\delta/4)^{\delta/5}} \qquad \Longrightarrow \qquad y_i^{t+q} \leq y_i^t \frac{(1+\delta/4)^{q\tilde{v}_{ij}^t}}{(1-\delta/4)^{q\delta/5}}$$

The inequality on the left follows from $\rho = 5/\delta$. Note that we have the $1/(1-\delta/4)^{\delta/5}$ term because we decrease the weight of the unmatched vertices and then renormalize the total weight to α. The renormalization effectively increases the weight of the matched vertices by the same factor. The equations can be derived by first computing $\mathbf{u}^{t+1}$ and renormalizing.

For any $\delta > 0$ and $q \leq 1/\delta$, we have $(1+\delta/4)^q \leq (1+\delta/4)^{1/\delta} < e^{1/4} < 2$ and $1/(1-\delta/4)^{q\delta/5} \leq 1/(1-\delta/4)^{1/5} \leq 1+\delta$. Therefore,

$$y_i^{t+q} + y_j^{t+q} \leq (y_i^t + y_j^t) \frac{(1+\delta/4)^{q\tilde{v}_{ij}^t}}{(1-\delta/4)^{q\delta/5}} \leq (y_i^t + y_j^t) 2^{\tilde{v}_{ij}^t} (1+\delta)$$

Since $2^{\tilde{v}_{ij}^t} \leq 1 + \tilde{v}_{ij}^t$ for $0 \leq \tilde{v}_{ij}^t \leq 1$ and $(y_i^t + y_j^t) = w_{ij}(1 - \tilde{v}_{ij}^t)$ we have:

$$y_i^{t+q} + y_j^{t+q} \leq w_{ij}(1 - \tilde{v}_{ij}^t)(1 + \tilde{v}_{ij}^t)(1 + \delta) \leq w_{ij}(1 + \delta)$$

This implies that $\frac{y_i^{t+q} + y_j^{t+q} - w_{ij}}{w_{ij}} \leq \delta$ for all $y_{ij} \neq 0$ and therefore $\mathbf{M}_q(\mathcal{D}^{t+q}, \mathbf{x}) \leq \delta$. Now we can claim using induction that $\mathbf{x}$ remains admissible for all $q \leq 1/\delta$ iterations (the inductive hypothesis was necessary to ensure that α did not change). The lemma follows. □

Algorithm 20 Overall Algorithm for MWM.

1: In one pass, find a 6 approximate maximum matching using [43] and let α_0 be the weight of the matching. Also ensure $\alpha_0 \geq w_{ij}$ for all $(i, j) \in E$.

2: $u_i^1 = 1$ for all $i \in [n]$ and $\alpha = 6\alpha_0$

3: **for** $t = 1$ **to** T **do**

4: Given u_i^t, run the oracle (Algorithm 19).

5: If the oracle failed decrease α by factor $(1 + \frac{\varepsilon}{3})$ and repeat line 4.

6: Let $\mathbf{M}(i, \mathbf{x}^t) = \mathbf{A}_i \mathbf{y}^t - b_i$. ($\mathbf{x}$ is an admissible primal witness now)

7: $$u_i^{t+1} = \begin{cases} u_i^t (1 + \epsilon)^{\mathbf{M}(i, \mathbf{x}^t)/5} & \text{if } \mathbf{M}(i, \mathbf{x}^t) \geq 0 \\ u_i^t (1 - \epsilon)^{-\mathbf{M}(i, \mathbf{x}^t)/5} & \text{if } \mathbf{M}(i, \mathbf{x}^t) < 0 \end{cases}$$
(This line is modified compared to Algorithm 3.)

8: **end for**

9: Output $\frac{1}{T} \frac{1}{1+4\delta} \sum_t \mathbf{y}^t$.

Theorem 6.1.17. *Theorem 6.1.12 holds with $T = O(\frac{1}{\varepsilon^2} \log n)$ (and with $T+1$ passes) using Algorithm 20.*

Proof. We use Lemma 6.1.16 repeatedly. We can compute $\mathbf{x}^1$ (using Algorithm 18 and Algorithm 19 as oracle) and use it for the next $\frac{1}{\delta}$ iterations. Observe that Lemma 6.1.16 shows that we can omit the oracle for $\frac{1}{\delta}$ iterations, so there can be no failure and α cannot change. Repeating the same argument we compute the witness only for every $\frac{1}{\delta}$ iterations. Observe that the overall algorithm simplifies to the description given in Algorithm 20.

Therefore we have $O(\frac{\delta}{\varepsilon^3}\log n) = O(\frac{1}{\varepsilon^2}\log n)$ actual computations of the dual witness, we have a $(1-\varepsilon)$ approximation, where $\delta = \varepsilon/30$. Computation of each $\mathbf{x}$ requires a pass. Note that we may need to repeat an iteration if the $\mathbf{y}$ was not admissible (as in Theorem 6.1.12) — but this only adds $O(\frac{1}{\varepsilon})$ iterations. The space requirement is $O(n(T+\log n))$ since we need to only remember the different $\mathbf{x}$ values we computed. □

6.1.3 Removing the Dependency on n

In this section, we present $(1-\varepsilon)$ approximation algorithms for bipartite MCM and MWM where the number of passes does not depend on the number of nodes. In each case we use the Algorithm 20 but we use a subgraph of the input graph and apply further analysis to bound the number of iterations T. Moreover, instead of starting from an initial state $u_i^1 = 1$ we will start the algorithm with different values of u_i^1.

We will also need to use the Theorem 2.3.2 instead of Corollary 2.3.3. Recall that the number of iterations of the multiplicative weights update method is

$O(\frac{1}{\delta^3}(\ln \max_i \frac{\Psi_i}{\Upsilon_i^1}))$ where $\Upsilon_i^t = u_i^t/(\sum_j u_j^t)$ and $\Psi_i = \max_t \Upsilon_i^t$. Of these, $\frac{1}{\delta}$ iterations can be performed in a single pass. In what follows, we will reduce or bound the $(\ln \max_i \frac{\Psi_i}{\Upsilon_i^1})$ term. The key observation we will use in this regard is the following lemma:

Lemma 6.1.18. *If $u_i^1 \geq w_{ij}$ for all $(i,j) \in E$ then during the execution of Algorithm 20, $y_i \leq 2u_i^1$.*

Proof. As the parameter α is decreased in Algorithm 20, (because a larger value of α ended up returning a primal feasible solution and so we are now decreasing α) the value of y_i decreases because $\mathbf{u}$ remains unchanged but α decreases. Thus it suffices to analyze the case when we do not change α.

We first observe that if $y_i^t \geq w_{ij}$ for all $(i,j) \in E$ then vertex i is not involved in any violations. Then no edge adjacent to i can be chosen in $\mathbf{x}$ and we will have $\mathbf{M}(i,\mathbf{x}) = \sum_{j:(i,j)\in E} x_{ij} - 1 = -1$. Then we will be setting $u_i^{t+1} = u_i^t(1-\epsilon)^{1/5}$ (based on the subroutine Algorithm 20). Moreover observe that $\sum_j u_j^{t+1} \geq \sum_j (1-\epsilon)^{1/5} u_j^t$, since for every j we have $\mathbf{M}(j,\mathbf{x}) \geq -1$. Therefore,

$$y_i^{t+1} = \frac{\alpha u_i^{t+1}}{\sum_j u_j^{t+1}} = \frac{\alpha u_i^t (1-\epsilon)^{1/5}}{\sum_j u_j^{t+1}} \leq \frac{\alpha u_i^t (1-\epsilon)^{1/5}}{\sum_j (1-\epsilon)^{1/5} u_j^t} = \frac{\alpha u_i^t}{\sum_j u_j^t} = y_i^t$$

Therefore y_i can increase only if it is involved in some violation. But then $y_i < w_{ij} \leq u_i^1$. So the maximum value y_i can achieve is when it is increased in a single step of Algorithm 20 to above u_i^1. The maximum value u_i^{t+1}/u_i^t in a step of Algorithm 20 is $(1+\epsilon)^{\mathbf{M}(i,\mathbf{x})/5} \leq (1+\epsilon)^{1/\delta}$. Note that $\epsilon \leq \frac{\delta}{4\ell}$ and therefore $(1+\epsilon)^{1/\delta} \leq e^{1/4}$. However we may also be decreasing $\sum_i u_i^t$; which can decrease by a factor $(1-\epsilon)^{1/5}$. Therefore

y_i, which is the relative contribution of u_i to $\sum_i u_i$ can increase at most by a factor of $e^{1/4}(1-\epsilon)^{-1/5}$ which is at most 2 for $\epsilon \leq \frac{1}{2}$. The lemma follows. $\square$

In Algorithm 20 $\Upsilon_i^t = y_i^t/\alpha$. Note $\alpha \geq OPT/6$ after the first pass where OPT is the maximum weighted matching. Therefore if $u_i^1 \geq w_{ij}$ for all $(i,j) \in E$ then using Lemma 6.1.18 $\Psi_i \leq \frac{12u_i^1}{OPT}$ and $\frac{\Psi_i}{\Upsilon_i^1} \leq \frac{12(\sum_j u_j^1)}{OPT} = O(\frac{(\sum_j u_j^1)}{OPT})$. Setting $\delta = \varepsilon/30$ we get a variant of Theorem 6.1.17 as follows:

Theorem 6.1.19. *If $w_{ij} \leq u_i^1$ for all edges $(i,j) \in E$ then for any $\varepsilon \leq \frac{1}{2}$ in T passes where $T = O(\frac{1}{\varepsilon^2} \log \frac{(\sum_j u_j^1)}{OPT})$, and $O(n(T + \log n))$ space we can compute a $(1-\varepsilon)$ approximation for maximum weighted matching in bipartite graphs.*

The Simple Case of MCM

In this context OPT denotes the size of the maximum cardinality matching in G. Consider the Algorithm 21 and the following lemma:

Lemma 6.1.20. *Let OPT_S denote the size of maximum matching in the subgraph induced by the vertex set $S \subseteq V$, then (using the notation of Algorithm 21), we have $OPT - OPT_{S_{t+2}} \leq \frac{2}{3}(OPT - OPT_{S_t})$. This proof does not use bipartiteness.*

Proof. Fix an optimal solution in the original graph G and an optimal solution in the subgraph induced by S_t. From the symmetric difference of two matchings, we can find $OPT - OPT_{S_t}$ vertex disjoint augmenting paths, say $\mathcal{P}$. We show that at least $\frac{1}{3}|\mathcal{P}|$ vertex disjoint augmenting paths are included in the graph induced by S_{t+2}.

Algorithm 21 A constant pass algorithm for maximum cardinality matching

1: Find a maximal matching and find a 2 approximation of OPT.

2: Let S_0 be the set of vertices that are matched.

3: **for** $t = 1$ **to** $2\left\lceil \frac{\log(2/\varepsilon')}{\log(3/2)} \right\rceil$ **do**

4: Find a maximal matching between S_{t-1} and $V - S_{t-1}$. Let T_t be the set of vertices in the maximal matching.

5: $S_t = S_{t-1} \cup T_t$.

6: **end for**

7: Let G' be a subgraph induced by S_T. This can be achieved by filtering the stream.

8: Run Algorithm 20 on G'

Order the vertex disjoint augmenting paths $\mathcal{P}$ arbitrarily. Let $i' - i - Z - j - j'$ be the first path (where Z is some sequence of vertices in S_t). Then $i', j' \notin S_t$ and $i' \neq j'$. In what follows we will show the condition $\mathcal{C}$: we have an augmentation path $i'' - i - Z - j - j''$ available in S_{t+2} for some $i'' \neq j'', i'', j'' \in V - S_t$ and $i'', j'' \in S_{t+2}$. (Note $\{i', j'\}$ can intersect $\{i'', j''\}$.)

If we prove this condition $\mathcal{C}$, then any augmentation path we find can remove at most two additional paths in $\mathcal{P}$ (since i'', j'' are now unavailable). Therefore we can find at least $\frac{1}{3}|\mathcal{P}|$ augmentations in S_{t+2}. This means that $OPT_{S_{t+2}} \geq OPT_{S_t} + \frac{1}{3}(OPT - OPT_{S_t})$ and therefore

$$OPT - OPT_{S_{t+2}} \leq OPT - \left(OPT_{S_t} + \frac{1}{3}(OPT - OPT_{S_t})\right)$$

Therefore if we prove this condition $\mathcal{C}$ the lemma follows. We now prove the condition

$\mathcal{C}$. If both i', j' were included in the matching in step $t+1$ or $t+2$ then the condition holds with $i'' = i'$ and $j'' = j'$ since we consider all edges in the induced subgraph. Therefore at least one of them, say i', was not included in any of the two maximal matching in steps $t+1, t+2$. This means that i was matched to some $\tilde{i}_1$ in step $t+1$ and some $\tilde{i}_2$ in step $t+2$ with $\tilde{i}_1 \neq \tilde{i}_2$. Now two cases arise (i) $j' \neq \tilde{i}_1$ and (ii) $j' = \tilde{i}_1$. In case (i) where $j' \neq \tilde{i}_1$, it must be that j or j' was matched in step $t+1$ since the edge (j, j') was available. If j' was matched then the condition is satisfied with $i'' = \tilde{i}_1$ and $j'' = j'$. Otherwise j was matched, say to j'', in step $t+1$ and $j'' \neq \tilde{i}_1$ since i is matched to $\tilde{i}_1$ in the same matching. The condition is satisfied with $i'' = \tilde{i}_1$. In case (ii) where $j' = \tilde{i}_1$ the condition is satisfied with $i'' = \tilde{i}_2$ and $j'' = j'$. Therefore the lemma follows. $\square$

If Lemma 6.1.20 is repeated as many times as in Algorithm 21, the difference between OPT (the size of the optimal matching in G) and the optimal solution in the subgraph G' is at most $\left(\frac{2}{3}\right)^{(\log \frac{2}{\varepsilon'})/(\log \frac{3}{2})} = 2^{-\log \frac{2}{\varepsilon'}} = \varepsilon'/2$ times OPT (notice that $OPT - OPT_{S_0} \leq OPT$ since we started with a 2 approximation). Therefore G' now contains a $(1-\varepsilon'/2)$ approximation of the maximum matching in G. The size of each maximal matching is $O(|OPT|)$ and we repeat $O(\log \frac{1}{\varepsilon})$, the subgraph contains at most $O(|OPT| \log \frac{1}{\varepsilon})$ vertices. The number of passes to find the subgraph is $O(\log \frac{1}{\varepsilon})$ since we can find a maximal matching in one pass. Using Theorem 6.1.19 we have:

Theorem 6.1.21. *For any $\varepsilon \leq \frac{1}{2}$ Algorithm 21 provides a $(1-\varepsilon)$ approximation for the maximum cardinality matching problem in bipartite graphs using $T =$*

$O(\frac{1}{\varepsilon^2}\log\log\frac{1}{\varepsilon})$ passes, and $O(n'(T+\log n'))$ space where $n' = \min\{n, |OPT|\log\frac{1}{\varepsilon}\}$. This implies a $\frac{2}{3}(1-\varepsilon)$ result for general graphs using the integrality gap results of [47, 48]. The size of the matching can be computed in $O(n'\log n')$ space.

Observe that to estimate the size we only need to remember G' and the $\mathbf{u}$ both of which can be done using $O(n')$ space. The oracle (Algorithm 19) requires $O(n'\log n')$ space.

The Not So Simple Case of MWM

The weighted case is significantly more difficult than the unweighted case. The subgraph will now be expressed implicitly using the vertex weights u_i as proxy. In the language of Linear Programming this means that, instead of starting from an uniform random sample of the constraints, we will start from a weighted sample. Let the maximum weighted matching be $\mathcal{M}$.

Before proceeding further, for the rest of this section we assume that the weights are discrete, i.e., $w_{ij} \in \{1, (1+\nu), \cdots, (1+\nu)^L\}$ where $L = O(\frac{1}{\nu}\log\frac{n}{\nu})$ to simplify the analysis for $\nu \leq 1/6$. Observe that if $\nu \geq 1/n$ then $L = O(\frac{1}{\nu}\log n)$. This discretization can be achieved in four steps and a single pass by: (i) using a single pass to find both a $\frac{1}{6}$ approximation of $w(\mathcal{M})$ using the algorithm of [43], and the maximum weight edge. Denote the larger of these two values by w' (which is a lower bound on the weight of $\mathcal{M}$). (ii) deciding to ignore all edges of weight $\nu w'/n$ and (iii) deciding to multiply all edge weights by $n/(\nu w')$ and (iv) performing the discretization by rounding down

the weights to powers of $(1+\nu)$. Note that in step (i) we reduce the optimal solution by a $(1-\nu)$ factor. Subsequently given any matching in this scaled setting, we have a matching in the original setting which is related by a simple scaling factor. The discretization of the weights reduces the optimal solution by another factor which is at most $1/(1+\nu)$.

Given a discretized set of edges we run Algorithm 22. The Algorithm 22 that computes the weights of the vertices is similar to Algorithm 21, but is significantly non-trivial. Let $\mathcal{M}'$ denote the maximum weight matching in this new discretized setting and its weight be $w(\mathcal{M}')$. We ensure that:

C1: $\sum_i u_i^1 \leq \left(\frac{42}{\nu^3} \ln \frac{1}{\nu}\right) w(\mathcal{M}')$.

C2: Let $G' = (V, E')$ be a subgraph that consists of edges (i, j) such that $w_{ij} \leq u_i^1, u_j^1$. Then, G' contains a matching with weight at least $(1-3\nu)w(\mathcal{M}')$.

Hence, using Theorem 6.1.19 we obtain a $(1-\varepsilon)$-approximation of the maximum matching $\mathcal{M}''$ in G' in $O(\frac{1}{\varepsilon^2} \log \frac{1}{\nu})$ passes. Then using C2 we would have a $(1-3\nu)(1-\varepsilon)$ approximation for the maximum weight matching $\mathcal{M}'$ in G'. This corresponds to a $(1-3\nu)(1-\varepsilon)\frac{(1-\nu)}{(1+\nu)}$ approximation for the maximum weight matching $\mathcal{M}$ in G — the weights in the original graph are scaled differently, but the relationship is one-to-one. Setting $\nu = \frac{\varepsilon'}{16}$ and $\varepsilon = \varepsilon'/2$ we would get a $(1-\varepsilon')$ approximation of $\mathcal{M}$ (for all $\varepsilon' \leq \frac{1}{2}$). We now proceed to ensure C1 and C2.

Lemma 6.1.22. *(Condition C1.)* $\sum_i u_i^1 \leq \left(\frac{42}{\nu^3} \ln \frac{1}{\nu}\right) w(\mathcal{M}')$. *Bipartiteness is not used in this proof.*

Algorithm 22 A constant pass algorithm for maximum weighted matching.

(i, j) is in **level** k if $w_{ij} = (1+\nu)^k$

for each level $k = 0, 1, \cdots, L$ **in parallel do**

  Find a maximal matching E_k^0.

  Let C_k be the set of nodes matched in the maximal matching.

  Let $S_k^1 = C_k$

  for $t = 1$ **to** $q = 8\lceil \frac{1}{\nu^2} \ln \frac{1}{\nu} \rceil$ **do**

    Find a maximal matching E_k^t between C_k and $V - S_k^t$.

    Let T_k^t be the set of nodes matched in the maximal matching.

    $S_k^{t+1} = S_k^t \cup T_k^t$.

  end for

  Let $\mathcal{N}(i, k)$ denote the neighbors of vertex i in $\cup_{t=0}^{q} E_k^t$.

end for

Let $u_i^1 = (1+\nu)^k$ for the maximum k with $i \in S_k^q$. Vertices not present in any S_k^q have $u_i^1 = 0$.

Let $G' = (V, E')$ where $E' = \{(i, j) : w_{ij} \leq u_i^1, u_j^1\}$.

Run Algorithm 20 on G' with initial weights u_i^1 and return its result.

Proof. For a vertex i define $k(i)$ to be the maximum k with $i \in S_k^q$. Therefore $u_i^1 = (1+\nu)^{k(i)}$. In what follows we will show a charging scheme where we charge u_i^1 to different edges in $\mathcal{M}'$. Consider the edge $e_i = (i,j)$ that caused the inclusion of i to $S_{k(i)}^q$. Note that $|\mathcal{N}(i',k')| \leq q$ for all i', k'.

At least one of i and j must be matched in $\mathcal{M}'$, otherwise $\mathcal{M}'$ is not optimal. Moreover either i or j must have an edge adjacent to it in $\mathcal{M}'$ with weight at least $\frac{1}{2}(1+\nu)^{k(i)}$ (otherwise we can remove both those edges and add e_i to increase the weight of $\mathcal{M}'$). Let the edge with the larger weight (between the two possible edges in $\mathcal{M}'$ adjacent to i,j) be $f(e_i)$. We charge $f(e_i)$ the value u_i^1. Note that $f(e_i), e_i$ are adjacent.

Now consider an edge $e = (i',j') \in \mathcal{M}'$ with $w_e = (1+\nu)^k$. This collects a charge for any vertex i in level $k(i)$ such that $\frac{1}{2}(1+\nu)^{k(i)} \leq (1+\nu)^k = w_e$. In each such level $k(i)$, we can have $e = f(e_i)$ for at most $2q+2$ different vertices i since e_i, e must be adjacent. If $e_i = (i',i)$ then there are at most $q+1$ possibilities for i (including i'). This is because either $i = i'$ or $i \in \mathcal{N}(i',k(i))$ and $|\mathcal{N}(i',k(i))| \leq q$. Counting the e_i that arise from j' as well, we know that $e = f(e_i)$ for at most $2q+2$ vertices i. Therefore the charge on e from vertices i with the largest value of $k(i)$ is $2(q+1)2w_e$. From the vertices that are in the immediately lower level, the charge is $\frac{4(q+1)w_e}{(1+\nu)}$. Summing over all the levels, the charge is

$$4(q+1)w_e\left(1+\frac{1}{1+\nu}+\frac{1}{(1+\nu)^2}+\cdots\right) \leq \frac{4(q+1)(1+\nu)}{\nu}w_e$$

Summing over all edges in $\mathcal{M}'$, since $\frac{4(q+1)(1+\nu)}{\nu} \leq \frac{42}{\nu^3}\ln\frac{1}{\nu}$ we have the desired result (we use the fact that $q+1 \leq \frac{9q}{8}$). $\square$

Lemma 6.1.23. *(Condition C2.) G' contains a matching with weight at least* $(1 - 3\nu)w(\mathcal{M}')$.

Proof. We start with $\mathcal{M}'$ and modify it into a matching $\mathcal{F}$ so that $\mathcal{F}$ contains only the edges in G'. We charge the loss induced by the modification to the edges in $\mathcal{M}' \cup \mathcal{F}$. We first describe the modification procedure:

1. Initially $M = \mathcal{M}'$. $\mathcal{F} = \emptyset$. We will maintain $M \cup \mathcal{F}$ to be a matching.

2. Pick the edge in M with the highest weight. Let this edge be $e = (i, j)$ with weight $w_e = (1 + \nu)^k$ in level k. Since E_k^0 was a maximal matching, either i or j is in C_k. Without loss of generality, let it be i. Thus $k(i) \geq k$. If $k(j) \geq k$ then both u_i^1, u_j^1 are at least w_e and $e \in G'$. We add (i, j) to $\mathcal{F}$ and remove (i, j) from M. Therefore it suffices to consider $k(j) < k$ and $j \notin S_k^q$ then i has at least q neighbors in S_k^q and $\mathcal{N}(i, k) = q$; since the edge (i, j) was available for potential inclusion in the q maximal matchings.

   (a) Suppose there is $i' \in \mathcal{N}(i, k)$ that is not matched in M or $\mathcal{F}$. Then $k(i') \geq k$ and $(i, i') \in G'$. Add (i, i') to $\mathcal{F}$ and remove (i, j) from M.

   (b) Otherwise each $i' \in \mathcal{N}(i, k)$ is matched in M or $\mathcal{F}$. If i' is matched in M denote its partner to be $\sigma(i', M)$. Otherwise i' is matched in $\mathcal{F}$ and denote its partner as $\sigma(i', \mathcal{F})$.

      i. If there exist at least $q/2$ vertices (q is even) in $\mathcal{N}(i, k)$ which are matched in $\mathcal{F}$, then delete (i, j) from M and give every $(i', \sigma(i', \mathcal{F}))$

where $i' \in \mathcal{N}(i,k)$ a **red** charge of $\frac{2}{q}w_e$.

ii. If there exists $i' \in \mathcal{N}(i,k)$ which is matched in M and its weight $w((i',\sigma(i',M))) < \nu w_e$, then we delete both (i,j) and $(i',\sigma(i',M))$ from M and add (i',i) to $\mathcal{F}$. Note $(i',i) \in G'$. Then (i',i) collects a **green** charge of $w((i',\sigma(i',M)))$.

iii. Otherwise, there exist at least $q/2$ vertices in $\mathcal{N}(i,k)$ which are matched in M, let this set be Q_i. Find the smallest weight edge in M incident on a vertex in Q_i, let that vertex be i_0. Delete both (i,j) and $(i_0,\sigma(i_0,M))$ from M and add (i,i_0) to $\mathcal{F}$. Each edge $(i',\sigma(i',M))$ where $i' \in \mathcal{N}(i,k)$ receives a **blue** charge of $\frac{2}{q}w_0$ where $w_0 = w(i_0,\sigma(i_0,M))$.

Observe that the sets $\mathcal{N}(i,k)$ are disjoint for a fixed k. This is because the matched vertices S_k^t are ruled out from participating in step $t+1$ or later. Observe that the edges are added to $\mathcal{F}$ in non-increasing order of weight. Moreover, during the execution of the above procedure, at any point we have the invariant $\mathcal{I}$: that every edge in $\mathcal{F}$ has a weight at least as much as the edge with the heaviest weight in M.

The **red** charges are collected by edges in $\mathcal{F}$. Consider edge $e \in \mathcal{F}$ with $w_e = (1+\nu)^k$. Edge e collects a **red** charge from edge e' if $w_{e'} \leq w_e$. This is a consequence of the invariant $\mathcal{I}$. Moreover, for each $k' \leq k$ the edge e can collect 2 such **red** charges for edges e'. This is because the two endpoints can be in $\mathcal{N}(i,k')$ for at most 2 different choices of i (this is a consequence of $\mathcal{N}(i,k')$ being disjoint for a fixed k).

The charge collected due to edges e' in level k' is $2\frac{2}{q}(1+\nu)^{k'}$. The total **red** charge collected by e can be bounded by $\sum_{k'=0}^{k} 2\frac{2}{q}(1+\nu)^{k'} \leq \frac{4}{\nu q}(1+\nu)^{k} = \frac{4}{\nu q}w_e$. The overall **red** charge sums to $\frac{4}{\nu q}w(\mathcal{F})$.

An edge $e \in \mathcal{F}$ collects a **green** charge from edge e' if $w_{e'} \leq \nu w_e$. Moreover, this charge is collected at most once. Therefore the total **green** charge is $\nu w(\mathcal{F})$.

The **blue** charges are collected by the edges in $\mathcal{M}'$. Consider edge $e \in \mathcal{M}'$ with $w_e = (1+\nu)^k$ which collect a **blue** charge when edge e' was the heaviest weight edge in M which was deleted from M along with e''. Observe that since we were considering the edges in M in decreasing order of weight, $w_{e'} \geq w_e$. Moreover $w_e \geq w_{e''}$, otherwise we would have deleted e and charged e'' in that step. And finally, $w_{e''} \geq \nu w_{e'}$, otherwise we would be in the green case. Therefore we have $w_e \geq \nu w_{e'}$ and the edge e is charged at most $\frac{2}{q}w_{e''} \leq \frac{2}{q}w_e$.

Let $w_{e'} = (1+\nu)^{k'}$ then $(1+\nu)^{k'} \geq (1+\nu)^{k} \geq \nu(1+\nu)^{k'}$ and we have $0 \leq k' - k \leq \frac{\ln\frac{1}{\nu}}{\ln(1+\nu)} \leq \frac{2}{\nu}\ln\frac{1}{\nu}$. The edge e can collect 2 such **blue** charges for edges e' in level k' (again follows from $\mathcal{N}(i,k')$ being disjoint). The total **blue** charge on edge e is at most $(\frac{2}{\nu}\ln\frac{1}{\nu})2\frac{2}{q}w_e = (\frac{8}{\nu q}\ln\frac{1}{\nu})w_e$. The overall **blue** charge sums to at most $\left(\frac{8}{\nu q}\ln\frac{1}{\nu}\right)w(\mathcal{M})$.

Observe that we maintained that $w(\mathcal{M}') = w(\mathcal{F}) + A$ where A is the total charge. Putting the charges together, we have

$$w(\mathcal{M}') \leq w(\mathcal{F}) + \frac{4}{\nu q}w(\mathcal{F}) + \nu w(\mathcal{F}) + \frac{8}{\nu q}\ln\frac{1}{\nu}w(\mathcal{M}')$$

Using $q = 8\lceil \frac{1}{\nu^2} \ln \frac{1}{\nu} \rceil$ and rearranging we get

$$(1-\nu)w(\mathcal{M}') \leq \left(1 + \frac{\nu}{2\ln\frac{1}{\nu}} + \nu\right) w(\mathcal{F}) \leq (1+2\nu)w(\mathcal{F})$$

This translated to $w(\mathcal{F}) \geq \frac{1-\nu}{1+2\nu} w(\mathcal{M}') \geq (1-3\nu)w(\mathcal{M}')$ for all $\nu \leq \frac{1}{6}$ and the lemma follows. □

It takes $q = O((\frac{1}{\varepsilon'})^2 \log \frac{1}{\varepsilon'})$ passes (for this setting of $\nu = \varepsilon/16$, see the discussion before Lemma 6.1.22) and $O(nL) = O(\frac{n \log n}{\varepsilon'})$ space to find the subgraph G'. Therefore (changing variables), we have

Theorem 6.1.24. *For any $\varepsilon \leq \frac{1}{2}$ in $T = O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ passes, and $O(n(T + \frac{\log n}{\varepsilon}))$ space we can compute a $(1-\varepsilon)$ approximation for the maximum weighted matching in bipartite graphs. This translates to a $\frac{2}{3}(1-\varepsilon)$ approximation for general graphs using the integrality gap results of [47, 48]. The weight can be estimated using $O(\frac{n}{\varepsilon} \log n)$ space.*

Observe that to estimate the weight we need to compute and store the subgraph G' which can be done using $O(\frac{n}{\varepsilon} \log n)$ space since we need to remember $O(n)$ vertices for each of the discretized weight levels. If we are only interested in the weight, the computation of the Algorithm 20 only needs $O(n \log n)$ space for the oracle and can remember $\mathbf{u}$ in space $O(n)$.

6.2 Approach II : Implicit Verification

In this section, we introduce the implicit verification approach to oracle design. Again, we use the context of the multiplicative weights update method (see Section 2.3.1) but the approach itself is applicable to the fractional packing/covering framework as well.

In Section 6.1, we prove the infeasibility of α by producing a dual feasible solution. However, the construction of a dual feasible solution is not always easy even if a dual candidate is given. The natural question arises: *is it possible to prove the infeasibility without producing an explicit dual solution?* The answer to the question is yes. Suppose that we have a 2-approximation algorithm for the problem and the approximation algorithm fails to produce a solution with objective value at least $\alpha/2$. This simply proves that α is infeasible. We apply this idea and obtain a general approach to construct an oracle. Algorithm 23 and Algorithm 24 describes such an approach.

Recall the goal of the oracle in Section 6.1. We want to find $\mathbf{x} \in \{\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \alpha; \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$. The primal-dual algorithm maintains $\mathbf{u}$ and then, the goal of the oracle is to find $\mathbf{x}$ such that $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \alpha$ and $\mathbf{u}^{\mathrm{T}}(\mathbf{A} - \mathbf{b}) \leq \delta \sum_j \mathbf{u}_j$. As a consequence, the problem reduces into a problem of finding x_i such that c_i is large and $\mathbf{A}_i^T\mathbf{u}$ is small.

Algorithm 23 describes the implicit verification approach. In Algorithm 23, the two conditions are summarized into "effective coefficients" in $\mathcal{L}(\mathbf{x}, \varrho) = \mathbf{c}^{\mathrm{T}}\mathbf{x} - \varrho\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ using the Lagrangian multiplier method. Note that we lose $(1 - \delta)$ factor in the

Algorithm 23 General Oracle Construction (Implicit).

Let $\mathcal{L}(\mathbf{x}, \varrho) = \mathbf{c}^{\mathrm{T}}\mathbf{x} - \varrho\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}$.

Let $\gamma = \mathbf{u}^{\mathrm{T}}\mathbf{b}$.

Consider:

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1-\delta)\alpha, \mathbf{u}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{x} \leq \mathbf{u}^{\mathrm{T}}\mathbf{b}, \mathbf{A}\mathbf{x} \leq \rho\mathbf{b} \tag{LP7}$$

$$\mathcal{L}(\mathbf{x}, \varrho) \geq \alpha - \varrho\gamma, \mathbf{A}\mathbf{x} \leq \rho\mathbf{b} \tag{LP8}$$

Let $\mathbf{x}^{\varrho}$ be a solution of LP8 for ϱ. *If α is feasible, we always find $\mathbf{x}$ that satisfies LP8 using Algorithm 24.*

Use $O(\frac{1}{\delta})$ (parallel) calls to LP8 to find ϱ^{+} and ϱ^{-} that satisfies:

1. $\varrho^{-} \leq \varrho^{+} \leq \varrho^{-} + \frac{\delta\alpha}{2\gamma}$,
2. $\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}^{\varrho^{+}} \leq \mathbf{u}^{\mathrm{T}}\mathbf{b}$, and
3. $\mathbf{c}^{\mathrm{T}}\mathbf{x}^{\varrho^{-}} \geq \alpha$.

if any call to LP8 returns $\mathbf{x}^{\varrho}$ such that $\mathcal{L}(\mathbf{x}^{\varrho}, \varrho) < \alpha - \varrho$ **then**

  Declare a failure. α is infeasible.

else

  Take a linear combination of $\mathbf{x}^{\varrho^{+}}$ and $\mathbf{x}^{\varrho^{-}}$ to find $\mathbf{x}$ that satisfies LP7.

  Return $\mathbf{x}$ as a primal witness.

end if

Algorithm 24 Algorithm for LP7.

1: *If α is feasible, there exists $\mathbf{x}$ such that:*

$$\mathcal{L}(\mathbf{x}, \varrho) \geq \alpha - \varrho\gamma, \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{LP9}$$

2: **if** $\alpha - \varrho\gamma < 0$ **then**

3: Return $\mathbf{x} = \mathbf{0}$ (a zero vector).

4: **end if**

5: Find a ρ-approximate solution $\mathbf{x}'$ for LP9, i.e.,

$$\mathcal{L}(\mathbf{x}', \varrho) \geq \frac{1}{\rho}(\alpha - \varrho\gamma), \mathbf{A}\mathbf{x}' \leq \mathbf{b}.$$

6: Return $\mathbf{x} = \rho\mathbf{x}'$.

objective value as a consequence, i.e., the linear combination of primal witnesses has an objective value of $(1-\delta)\alpha$ (or greater) rather than α. *The Lagrangian multiplier method reduces the problem into a problem of the same format but with a different objective function and relaxed constraints. (See LP8)* Algorithm 24 solves the reduced problem using a ρ-approximation algorithm.

Lemma 6.2.1 states that Algorithm 24 is always successful if α is feasible. The consequence of this lemma is that if the algorithm is unsuccessful, i.e., it returns $\mathbf{x}$ such that $\mathcal{L}(\mathbf{x}, \varrho) < \alpha - \varrho\gamma$, we can conclude that the guess α is infeasible, thus, the name implicit verification. Lemma 6.2.2 states that Algorithm 23 returns an admissible primal witness. Combined with Corollary 2.3.3, the overall algorithm finds a feasible primal solution $\mathbf{x}$ with objective value $(1-\delta)\min_i \frac{b_i}{b_i+\delta}\alpha$. Note that we have an additional factor $(1-\delta)$.

Lemma 6.2.1. *If $b_i > 0$ for all constraints and α is feasible, Algorithm 24* always *returns $\mathbf{x}$ that satisfies LP8 for any $\varrho \geq 0$.*

Proof. Suppose that LP9 is feasible. Then, the rest of the proof is straightforward. If $\alpha - \varrho\gamma$ is negative, it is obvious that the zero vector would satisfy LP8. Otherwise, the ρ-approximation algorithm will find $\mathbf{x}'$ that satisfies all the constraints of the original with the objective value at least $1/\rho$ times the optimal value which is greater or equal to $\frac{1}{\rho}(\alpha - \varrho\gamma)$. Therefore, the simple scaling step (Step 6) gives a desired result if $b_i > 0$ for all constraints.

Let $\mathbf{x}^{OPT}$ be the optimal solution. Then, $\mathbf{c}^{\mathrm{T}}\mathbf{x}^{OPT} \geq \alpha$ and $\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x} \leq \mathbf{u}^{\mathrm{T}}\mathbf{b}$ since $\mathbf{A}_i\mathbf{x} \leq b_i$ and $u_i, b_i > 0$ for all i. Therefore, for any ϱ, $\mathcal{L}(\mathbf{x}^{OPT}, \varrho) \geq \alpha - \varrho\gamma$. The lemma follows. □

Lemma 6.2.2. *If $b_i > 0$ for all constraints and α is feasible, Algorithm 23 returns an admissible primal witness.*

Proof. We already proved the statement in step 4. The remaining parts are: (i) we can find ϱ^+ and ϱ^- that satisfies the conditions in step 5 and (ii) such $\mathbf{x}^{\varrho^+}$ and $\mathbf{x}^{\varrho^-}$ produces an admissible primal witness.

Recall that Algorithm 24 returns a zero vector for any $\varrho > \frac{\alpha}{\gamma}$. The zero vector satisfies the condition 2. On the other hand, $\mathbf{x}^0$ satisfies the condition 3 since $\mathcal{L}(\mathbf{x}, 0) = \mathbf{c}^{\mathrm{T}}\mathbf{x}$ and $\alpha - 0 \cdot \gamma = \alpha$. If we call LP8 for $\varrho = 0, \frac{\delta\alpha}{2\gamma}, \frac{\delta\alpha}{\gamma}, \cdots$, we will find ϱ^+ and ϱ^- that satisfies all three conditions with $O(\frac{1}{\delta})$ calls.

We take a linear combination $\mathbf{x} = a\mathbf{x}^{\varrho^+} + (1-a)\mathbf{x}^{\varrho^-}$, $a \in [0,1]$ such that $\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \gamma$. Note that

$$\begin{aligned} a\mathcal{L}(\mathbf{x}^{\varrho^+}, \varrho^+) + (1-a)\mathcal{L}(\mathbf{x}^{\varrho^-}, \varrho^-) &\geq \alpha - \varrho^-\gamma - a(\varrho^+ - \varrho^-)\gamma \\ &\geq (1-\delta)\alpha - \varrho^-\gamma \end{aligned}$$

where we use $\varrho^+ - \varrho^- \leq \frac{\delta\alpha}{2\gamma}$, $\gamma \geq 0$. Therefore

$$\begin{aligned} \mathbf{c}^{\mathrm{T}}\mathbf{x} &= a\mathcal{L}(\mathbf{x}^{\varrho^+}, \varrho^+) + (1-a)\mathcal{L}(\mathbf{x}^{\varrho^-}, \varrho^-) + a\varrho^+\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}^{\varrho^+} + (1-a)\varrho^-\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}^{\varrho^-} \\ &\geq (1-\delta)\alpha - \varrho^-\gamma + a(\varrho^+ - \varrho^-)\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}^{\varrho^+} + \mathbf{u}^{\mathrm{T}}\mathbf{A}(a\varrho^-\mathbf{x}^{\varrho^+} + (1-a)\varrho^-\mathbf{x}^{\varrho^-}) \\ &\geq (1-\delta)\alpha - \varrho^-\gamma + 0 + \mathbf{u}^{\mathrm{T}}\mathbf{A}(a\varrho^-\mathbf{x}^{\varrho^+} + (1-a)\varrho^-\mathbf{x}^{\varrho^-}) \\ &\qquad\qquad (\text{Using } \mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}^{\varrho^+} \geq 0 \text{ and } \varrho^+ - \varrho^- \geq 0) \\ &\geq (1-r)\alpha - \varrho^-\gamma + \mathbf{u}^{\mathrm{T}}\mathbf{A}\varrho^-\mathbf{x} \qquad (\text{Using } \mathbf{x} = a\mathbf{x}^{\varrho^+} + (1-a)\mathbf{x}^{\varrho^-}) \\ &\geq (1-r)\alpha - \varrho^-(\gamma - \mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}) = (1-r)\alpha \qquad (\text{Using } \gamma = \mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{x}) \end{aligned}$$

The lemma follows. □

The following lemma is a generalized version of the Lagrangian method we use here and will be used later in this thesis. We skip the proof since it is almost identical to the proof of Lemma 6.2.2. The only difference is that the "effective objective" does not have to be greater than $\beta_1 - \varrho\beta_2$. Rather an approximation is sufficient. Note that $\mathbf{A}_1$ and $\mathbf{A}_2$ are row vectors not matrices.

Lemma 6.2.3. *Suppose $\mathcal{P}$ is a polytope such that $\mathcal{P} \subset \{\mathbf{x}; \mathbf{A}_2\mathbf{x} \geq \mathbf{0}; \mathbf{x} \geq \mathbf{0}\}$ and*

$\beta_1, \beta_2 \geq 0$. *Consider the system LP10:*

$$\{\mathbf{x}; \mathbf{A}_1\mathbf{x} \geq \beta_1; \mathbf{A}_2\mathbf{x} \leq \beta_2; \mathbf{x} \in \mathcal{P}\} \tag{LP10}$$

For any $r > 0, 1 \geq q > 0$, given LP10 and a subroutine that finds $\mathbf{x} \in \mathcal{P}$ such that $(\mathbf{A}_1 - \varrho\mathbf{A}_2)\mathbf{x} \geq q(\beta_1 - \varrho\beta_2)$ for any $\varrho \geq 0$, we can find $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{A}_1\mathbf{x} \geq q(1-r)\beta_1$ and $\mathbf{A}_2\mathbf{x} \leq \beta_2$ with $O(\frac{1}{r})$ (parallel) invocations of the subroutine.

6.2.1 $O(\frac{1}{\varepsilon^2}\log n)$-pass Algorithm

In this section, we demonstrate the implicit verification approach using the bipartite MWM. A disadvantage of the approach is that we do not have a precise control over the primal witness since we use an approximation algorithm as a black box. As a consequence techniques in Sections 6.1.2 and 6.1.3 do not apply and we obtain a worse number of passes for the bipartite MCM and MWM.

However, the implicit verification approach allows us a easier oracle construction as long as we have a constant factor approximation algorithm. Therefore, it suits better in dealing with more complex LPs. For MWM, Feigenbaum et al. gave a one-pass 6-approximation algorithm in the insertion-only model [43]. [2]

Algorithm 25 and Algorithm 26 are the oracle for the bipartite MWM problem (LP5). Note that unlike the explicit verification approach, we do not need to design a new algorithm for this specific purpose. We just use Algorithm 23 and Algorithm

[2]There are other one-pass algorithms that achieve a better approximation factor [95, 42]. However, any constant factor approximation would be sufficient to demonstrate the approach.

Algorithm 25 Oracle for the bipartite MWM.

Let $\mathcal{L}(\mathbf{x}, \varrho) = \sum_{(i,j)\in E} w_{ij}x_{ij} - \varrho \sum_i u_i^t \sum_j x_{ij}$.

Let $\gamma = \sum_i u_i^t$.

Consider:

$$\sum_{(i,j)\in E} w_{ij}x_{ij} \geq (1-\delta)\alpha, \sum_i u_i^t \sum_j x_{ij} \leq \sum_i u_i^t, \sum_i \sum_j x_{ij} \leq 6 \tag{LP11}$$

$$\mathcal{L}(\mathbf{x}, \varrho) \geq \alpha - \varrho\gamma, \sum_i \sum_j x_{ij} \leq 6 \tag{LP12}$$

Use $O(\frac{1}{\delta})$ parallel calls to LP12 to find ϱ^+, ϱ^- such that:

1. $\varrho^- \leq \varrho^+ \leq \varrho^- + \frac{\delta\alpha}{2\gamma}$,
2. $\sum_i u_i^t \sum_j x_{ij} \leq \sum_i u_i^t$, and
3. $\sum_{(i,j)\in E} w_{ij}\mathbf{x}_{ij}^{\varrho^-} \geq \alpha$.

if any call to LP12 returns $\mathbf{x}^\varrho$ such that $\mathcal{L}(\mathbf{x}^\varrho, \varrho) < \alpha - \varrho$ **then**

  Declare a failure. α is infeasible.

else

  Take a linear combination of $\mathbf{x}^{\varrho^+}$ and $\mathbf{x}^{\varrho^-}$ to find $\mathbf{x}$ that satisfies LP11.

  Return $\mathbf{x}$ as a primal witness.

end if

.

Algorithm 26 Subroutine for MWM.

1: *If α is feasible, there exists $\mathbf{x}$ such that:*

$$\sum_{i,j} x_{ij}(w_{ij} - \varrho(u_i^t + u_j^t)) \geq \alpha - \varrho\gamma$$

2: **if** $\alpha - \varrho\gamma < 0$ **then**

3: Return $\mathbf{x} = \mathbf{0}$ (a zero vector).

4: **end if**

5: In one pass, find a 6-approximate matching given "effective weight" $w'_{ij} = w_{ij} - \varrho(u_i^t + u_j^t)$ for $(i, j) \in E$.

6: Return $x_{ij} = 6$ for any (i, j) in the matching and $x_{ij} = 0$ for other edges.

24 with the 6-approximation algorithm as a subroutine. The width parameter of the oracle is 6.

Note that the space-reducing technique in Section 6.1.2 applies to this approach. The oracle uses $O(\frac{n}{\varepsilon})$ space for the computation and returns a vector with $O(n)$ non-zero coordinates. Overall, we obtain the following result using this oracle.

Theorem 6.2.4. *For any $\varepsilon \leq \frac{1}{2}$ in $T = O(\frac{1}{\varepsilon^2}\log n)$ passes, and $O(n(T + \frac{1}{\varepsilon}))$ space we can compute a $(1-\varepsilon)$ approximation for maximum weighted matching in bipartite graphs (the size can be computed in less space).*

6.3 Approach III: Dual-Primal Approach

In this section, we introduce the dual-primal approach. We use the context of the fractional packing/covering framework (see Section 2.3.2) rather than the multiplicative weights update method. The fractional packing/covering framework allows us to set aside some constraints as long as the primal witnesses satisfy those constraints in every iteration. These constraints were represented as a polytope $\mathcal{P}$. It is possible to modify the multiplicative weights update method to handle such constraints as well but it requires the modification of the standard form.

In Sections 6.1 and 6.2, we introduced two approaches to design an oracle. In this section, we present another approach of using the primal-dual algorithms. However, the approach is not about oracle design. While we presented a variety of techniques to utilize the primal-dual algorithms in the semi-streaming model, an obvious question remains:

Should we focus on the primal or on the dual?

Primal and dual LPs describe the same combinatorial object and are often interchangeable. But that does not hold under two circumstances: the presence of (i) approximations and (ii) resource constraints, specially in access to data. This immediately suggests that we have a choice between starting from the primal (the algorithm provides dual candidate to an oracle) or starting from the dual (the algorithm provides dual of dual, i.e., primal candidate, to an oracle). It would probably be more appropriate to call the latter style of algorithms as "dual-primal" since they would

start from the dual formulation. It is worth noting that in the context of approximation algorithms, "primal-dual algorithms" start from the dual formulation — they construct a feasible (possibly fractional) dual solution and an integral primal solution that relates directly to the feasible dual solution, see [92, 87]. It can be argued that a primal-dual approximation algorithm uses the structure of the dual polytope more critically. The structure of the dual polytope has also received a lot of attention in the exact optimization literature as well and notions such as total dual integrality (TDI) have been studied widely. Therefore it is reasonable to speculate if the benefits of studying the dual formulation also apply to fast solutions of LPs.

Suppose we want to solve a maximization problem $\{\max \mathbf{b}^T\mathbf{x}; \mathbf{A}^T\mathbf{x} \leq \mathbf{c}; \mathbf{x} \geq 0\}$. Say $\mathbf{A}, \mathbf{b}, \mathbf{c} \geq 0$. The dual is a minimization problem and we can conceive a feasibility version of $\exists \mathbf{y}$ such that $\mathcal{P}'(\beta) = \{\mathbf{A}\mathbf{y} \geq \mathbf{b}; \mathbf{c}^T\mathbf{y} \leq \beta; \mathbf{y} \geq 0\}$. To find such an $\mathbf{y}$ using the primal-dual paradigm we would repeatedly need to find some $\mathbf{y}'$ such that $\mathbf{u}^T\mathbf{A}\mathbf{y}' \geq (1 - O(\epsilon))\mathbf{u}^T\mathbf{b}$ where the $\mathbf{u}$ corresponds to weights on the constraints (rows of $\mathbf{A}$) for that iteration.

In the explicit verification approach, we prove that this polytope $\mathcal{P}'(\beta) = \{\mathbf{A}\mathbf{y} \geq \mathbf{b}; \mathbf{c}^T\mathbf{y} \leq \beta; \mathbf{y} \geq 0\}$ is infeasible through weak duality and a dual feasible solution that violates the $\mathbf{c}^T\mathbf{y} \leq \beta$ part; that is, provide $\mathbf{x}$ such that $\mathbf{b}^T\mathbf{x} > \beta$ and $\mathbf{A}^T\mathbf{x} \leq \mathbf{c}$. But notice that the proof of infeasibility is already a feasible solution for the problem we wanted to solve! Now simply consider staring from a small value of β which is provably impossible for the dual and slowly increase it by factors of $(1 + \epsilon)$. At some

value of β^* we will not be able to prove that the dual problem is infeasible – but then for $\beta^*/(1+\epsilon)$ we have a feasible solution for our original problem. Observe that we never have to worry about part (b) at all! A similar logic holds if we show $\mathbf{b}^T\mathbf{x} > (1-O(\epsilon))\beta$ and $\mathbf{A}^T\mathbf{x} \leq \mathbf{c}$. Observe that an amazing consequence also falls out from this discussion: *the actual* $\mathbf{y}_1, \mathbf{y}_2, \ldots$ *returned by the oracle are not so critical.* The only use of these $\mathbf{y}$ are in providing a "algorithmic proof" of the final β^* for which the dual-primal algorithm succeeded! This implies that we can seek a feasible solution of some $\mathcal{P}''(\beta)$ as long as we have a proof that the feasibility of $\mathcal{P}''(\beta)$ implies the feasibility of $\mathcal{P}'(\beta)$. Working with $\mathcal{P}''(\beta)$ allows us to prove faster converge bounds.

So the dual-primal paradigm starts from a parameterized polytope $\mathcal{P}''(\beta)$ which is somewhat related to the feasible dual of the optimization problem P we are trying to solve. The oracle either provides us a solution of P of value β, or eventually finds a β such that $\mathcal{P}''(\beta)$ is feasible. If we were increasing β slowly in powers of $(1+\epsilon)$ we would eventually find a β^* such that (i) $\mathcal{P}''(\beta^*)$ is feasible and therefore $\mathcal{P}'(\beta^*)$ is feasible and we immediately know that the optimum solution is at most β^* and (ii) we have a solution of P with value $(1-O(\epsilon))\beta^*$. $\mathcal{P}''$ is chosen so that the width (convergence) is lower (faster) and easier to analyze. The main hurdle is to produce the oracle which provides a clean proof of dual infeasibility as mentioned

The basic outline of the dual-primal method is outlined above. The natural question is how does the idea help? Does it help us prove some new result? Does it help us improve existing results? In the subsequent section, we show that both these

questions can be answered in the affirmative using bipartite MWM as an exemplar.

6.3.1 $O(n^{1+1/p})$-space $O(p/\varepsilon)$-pass Algorithm

In this section we prove the following:

Theorem 6.3.1. *For any* $0 < \epsilon \leq 1$ *and* $p \leq \frac{\log n}{64(\log \frac{1}{\epsilon} + \log\log n)}$ *we can construct an integral solution for the bipartite maximum weighted matching to a factor* $(1 - O(\epsilon))$ *in* $O(p/\epsilon)$ *passes and* $O(pm\,\mathrm{poly}(\epsilon^{-1}, \log n))$ *time and* $O(n^{1+1/p})$ *space.*

Recall that LP5 captures optimum matching in bipartite graphs (compare LP22). For $(i,j) \notin E$ we force $x_{ij} = 0$ and we use an undirected formulation; $x_{ij} = x_{ji}$.

$$\begin{aligned} \beta^* = \max \quad & \textstyle\sum_{(i,j)\in E} w_{ij} x_{ij} \\ \text{s.t} \quad & \textstyle\sum_{j:(i,j)\in E} x_{ij} \leq 1 \quad \forall i \in V \\ & x_{ij} \geq 0 \qquad\qquad \forall (i,j) \in E \end{aligned} \tag{LP5}$$

Definition 6.3.2. Assume that we the weights are rounded down to powers of $1/(1-\epsilon)$ and scaled such that the smallest weight is 1. Therefore any matching loses at most a factor of $(1-\epsilon)$. Given (i,j,w_{ij}) we can compute the "level" k of the edge (i,j) (higher level corresponds to higher weight). Let w_k be the weight of the edges in level

k. LP5 rewrites to LP13 with $\hat{\beta} \leq \beta^* \leq \hat{\beta}/(1-\epsilon)$. Note $x_{ijk} = x_{jik}$.

$$\begin{aligned}
\hat{\beta} = \max \sum_i \sum_k \sum_{(i,j)\in \text{level } k} & w_k x_{ijk} \\
\textstyle\sum_{(i,j)\in \text{level } k} x_{ijk} \leq x_{i(k)} & \quad \forall i,k \\
\textstyle\sum_k x_{i(k)} \leq 1 & \quad \forall i \\
x_{ijk}, x_{i(k)} \geq 0 & \quad \forall i,j,k
\end{aligned} \tag{LP13}$$

$$\begin{aligned}
\hat{\beta} = \min \textstyle\sum_i y_i & \\
y_{i(k)} + y_{j(k)} \geq w_k & \quad \forall i,j,k \\
y_i - y_{i(k)} \geq 0 & \quad \forall i,k \\
y_i, y_{i(k)} \geq 0 &
\end{aligned} \tag{LP14}$$

LP14 is the dual of LP13. We insert $y_{i(k)} \leq \Lambda w_k$ for $\Lambda \geq 1$ without affecting the optimality of LP13 – this defines the covering problem LP15. To solve LP15, we need to solve LP16 (Define $u_{ijk} = 0$ if $(i,j) \notin$ level k for convenience; note that $u_{ijk} = u_{jik}$. We also added the redundant constraint $y_{i(k)} \leq 4w_k/\epsilon$.):

$$\mathbf{A} : \left\{ y_{i(k)} + y_{j(k)} \geq w_k \quad \forall i,j,(i,j) \in \text{level } k \right.$$

$$\mathbf{P}(\beta,\Lambda) : \begin{cases} \textstyle\sum_i y_i \leq \beta & \\ y_i - y_{i(k)} \geq 0 & \forall i,k \\ y_{i(k)} \leq \Lambda w_k & \forall i,k \\ y_i, y_{i(k)} \geq 0 & \end{cases} \tag{LP15}$$

$$\mathbf{A}'(\Lambda) : \left\{ y_{i(k)} \leq \Lambda w_k \quad \forall i,k \right.$$

$$\sum_{(i,j),k} y_{i(k)} u_{ijk} \geq \left(1 - \frac{\epsilon}{2}\right) \sum_{(i,j),k} u_{ijk}$$

$$\tilde{\mathbf{P}}(\beta) : \begin{cases} \sum_i y_i \leq \beta & \\ y_{i(k)} \leq 4w_k/\epsilon & \forall i,k \\ y_i - y_{i(k)} \geq 0 & \forall i,k \\ y_i, y_{i(k)} \geq 0 & \end{cases} \tag{LP16}$$

The Basic Algorithm: Suppose that we have a solution of value $\sum_i y_i = \beta_0 < \hat{\beta}$ which satisfies $x_{i(k)} + x_{j(k)} \geq w_k/8$ and satisfies every other constraint of LP15 with $\Lambda = 1$ (such a solution is not difficult and discussed in Section 6.3.3). Now suppose we apply the fractional covering approach to LP15 with $\Lambda = 2$ (thus the $\Lambda = 1$ initial

solution also applies). The next lemma proves the framework.

Lemma 6.3.3. *Suppose that an oracle provides us a either (i) solution of LP16 with* $\Lambda = 2$ *(ii) a (compressed) feasible solution of LP5 such that* $\sum_{(i,j)} w_{ij}x_{ij} \geq (1 - \frac{\epsilon}{4})\beta$*, and in case (ii) we raise* β*. After* $O(\epsilon^{-2}\log n + \frac{1}{\epsilon})$ *queries to the oracle we have a* $(1 - 6\epsilon)$ *approximate solution to LP5. The weights* u_{ijk} *lie between* $n^{\pm O(1/\epsilon)}$*.*

Proof. Note that since $\mathbf{P}(\beta, \Lambda)$ is laminar as β increases; the solutions provided to LP16 for smaller values of β remain feasible. Note that we can keep reusing those solutions and eventually have $O(\epsilon^{-2}\log n)$ successful solutions to LP16 — since we cannot continue increasing β if $(1 - \epsilon)\beta > \beta^*$ because there cannot be be any such $\{y_{ij}\}$ (duality). Then for the current value of β we will have a $(1 - 3\epsilon)$ approximate solution for LP15 with $\Lambda = 2$. But this will imply that $\hat{\beta} \leq \beta/(1 - 3\epsilon)$ based on a simple scaling argument and duality. From the $\{x_{ij}\}$ corresponding to the previous β we have $\sum_{(i,j)} w_{ij}x_{ij} \geq (1 - \frac{\epsilon}{4})\beta/(1 + \epsilon)$. The bound on the weights follows from $(1 \pm \epsilon)^{O(\epsilon^{-2}\log n)}$. The lemma follows. □

A Modified Algorithm: The question therefore is to design an oracle as specified in Lemma 6.3.3. Consider a deferred estimation problem: we are given $\{\varsigma_{ij}\}$ with the promise that $\varsigma_{ij}/\chi \leq u_{ijk} \leq \varsigma_{ij}\chi$. We decide to store some of the edges based on ς_{ij}. Then the exact weights u_{ijk} of only those stored edges are revealed to us. We now have to produce nonnegative $\{u^s_{ijk}\}$ (obeying the symmetry $u^s_{ijk} = u^s_{jik}$) such that $(1 - \xi)\sum_j u^s_{ijk} \leq \sum_j u_{ijk} \leq (1 + \xi)\sum_j u^s_{ijk}$ for all i, k. If such a deferred estimation were available to us, we can immediately set $\chi = n^{1/(4p)}, \xi = \epsilon/8$ and reduce the

effective number of iterations by a factor of $t = \log_{1+\epsilon} \chi = O(\frac{1}{\epsilon}\frac{1}{p}\log n)$ to $O(p/\epsilon)$ from $O(\epsilon^{-2}\log n)$ if we solve LP17:

$$\sum_{(i,j),k} y_{i(k)} u^s_{ijk} \geq \left(1 - \frac{\epsilon}{8}\right) \sum_{(i,j),k} u^s_{ijk} \quad \text{in addition to } \mathbf{A}'(\Lambda) \text{ and } \tilde{\mathbf{P}}(\beta) \tag{LP17}$$

since easy calculations show that LP17 $\implies$ LP16. On the other hand, if LP15 with $\Lambda = 1$ is feasible then LP17 is feasible for $\Lambda = 1$ for any nonnegative $\{u^s_{ijk}\}$. This immediately suggests that the oracle is solving the fractional packing problem LP18 to an approximation $(1 + 6\delta)$ for $\delta = 1/6$ (see the review in Section 2.3.2).

$$\mathbf{A}'(1) : \{ y_{i(k)} \leq w_k \qquad \forall i, k \tag{LP18}$$

$$\sum_{(i,j),k} y_{i(k)} u^s_{ijk} \geq \left(1 - \frac{\epsilon}{8}\right) \sum_{(i,j),k} u^s_{ijk}$$

$$y_i, y_{i(k)} \in \tilde{\mathbf{P}}(\beta)$$

$$\sum_{i,k} \zeta_{ik} y_{i(k)} \leq \frac{13}{12} \sum_{i,k} \zeta_{ik} w_k \tag{LP19}$$

$$\sum_{(i,j),k} y_{i(k)} u^s_{ijk} \geq \left(1 - \frac{\epsilon}{8}\right) \sum_{(i,j),k} u^s_{ijk}$$

$$y_i, y_{i(k)} \in \tilde{\mathbf{P}}(\beta)$$

Observe that no initial solution is required and we need a MiniOracle to either (i) solution to LP19 given $\{\zeta_{ik}\}$ or (ii) a (compressed) feasible solution of LP5 such that $\sum_{(i,j)} w_{ij} x_{ij} \geq (1 - \frac{\epsilon}{4})\beta$. The fractional packing framework may declare LP18 to be infeasible, but that implies LP15 is infeasible for $\Lambda = 1$. Therefore the Oracle for Lemma 6.3.3 is achieved!

Before we proceed further, let us establish an algorithm for the deferred estimation problem:

Definition 6.3.4 (The Deferred Degree Counter Problem)**.** Consider the problem: We are given a weighted graph $G = (V, E, u)$ but the weights are *not revealed* to us. Instead we are given $\{\varsigma_{ij}\}$ with the promise that $\varsigma_{ij}/\chi \leq u_{ij} \leq \varsigma_{ij}\chi$. We can have

a single pass over the $m = |E|$ edges and have to produce a smaller summary data structure $\mathcal{D}$ (ξ-*deferred degree counter*) storing only some of the edges. Once the data structure is constructed, then the exact weights u_{ij} of only those edges stored in our structure $\mathcal{D}$ are revealed to us. We then have to output *a* ξ-*degree counter*, a graph $H = (V, E', u^s)$ such that $(1-\xi)\sum_j u_{ij} \leq \sum_j u^s_{ij} \leq (1+\xi)\sum_j u_{ij}$. Note that this is implemented in a single pass over the graph.

Lemma 6.3.5. (Proved in Section 6.3.2.) *Given bound ϱ and $\{\varsigma_{ij}\}$ factor and edge weights are in $[1, O(\operatorname{poly} n)]$, we can generate a deferred degree counter of size $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$. We can construct a graph $H = (V, E', u^s)$ such that $(1-\xi)\sum_j u_{ij} \leq \sum_j u^s_{ij} \leq (1+\xi)\sum_j u_{ij}$ when the edge weights of the stored edges are revealed. The algorithm runs in $O(\frac{\chi^2 m}{\xi^2}\log n)$ time and $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$ space. In this paper $\chi = n^{1/(4p)}$.*

We will be using Lemma 6.3.5 when $\chi = n^{1/(4p)}$ and $\xi = \epsilon/16$. To implement the MiniOracle we need two more steps. A generalization of Lemma 6.3.6 is proved in Section 9.1.

Lemma 6.3.6. *We can solve LP19 using $O(\log\frac{1}{\epsilon})$ invocations of LP20 where $0 \leq \varrho < \varrho_o = \frac{12\sum_{i,j,k} w_k u^s_{ijk}}{13\sum_{ik} w_k \zeta_{ik}}$.*

$$\sum_{i,k} y_{i(k)}\left(\sum_j u^s_{ijk} - \varrho\zeta_{ik}\right) + \varrho\sum_{i,k} w_k\zeta_{ik} \geq \sum_{(i,j),k} w_k u^s_{ijk} - \sum_{i,k} w_k\varrho\zeta_{ik}$$

$$\textit{subject to } \tilde{\mathbf{P}}(\beta) \tag{LP20}$$

Proof. In case (ii), we can stop the whole algorithm since we already obtained the fractional solution. So in the rest of this proof, we assume case (i). Suppose that the

main oracle returns $\mathbf{y}$ for $\varrho = 0$ that satisfies

$$\sum_{i,k} \zeta_{ik} y_{i(k)} \leq \frac{13}{12} \sum_{i,k} \zeta_{ik} w_k. \tag{6.3.1}$$

If $\mathbf{y}$ satisfies LP19, then $\mathbf{y}$ is the desired output.

Suppose not. For $\varrho \geq \varrho_0$, $\mathbf{y} = \mathbf{0}$ is a feasible solution which satisfies Equation 6.3.1. Now using a binary search over $[0, \varrho_0]$ and find an interval $[\varrho_1, \varrho_2]$ where the corresponding $\mathbf{y}_1$ does not satisfy Equation 6.3.1 while $\mathbf{y}_2$ satisfies Equation 6.3.1 (but both satisfy LP20). We can narrow the interval so that $\varrho_2 - \varrho_1 \leq \epsilon\varrho_0$, $\varrho_2 > \varrho_1$ in $O(\log \frac{1}{\epsilon})$ invocations of LP20. Now find a linear combination of $\mathbf{y}_1$ and $\mathbf{y}_2$ such that it exactly satisfies 6.3.1, i.e.,

$$\sum_{i,k} \zeta_{ik} y_{i(k)} = \frac{13}{12} \sum_{i,k} \zeta_{ik} w_k.$$

Then, such $\mathbf{y}$ satisfies LP19. Note that the factor $(1 - \frac{\epsilon}{8})$ in LP19 is introduced at this step and we can make it arbitrarily small by decreasing the length of the interval $[\varrho_1, \varrho_2]$. The details of the mathematical equations will be given in Section 9.1 where we give a generalization of Lemma 6.3.6 (see Lemma 9.1.6). □

Lemma 6.3.7. (The main oracle.) *Given any nonnegative $\{u^s_{ijk}\}, \{\zeta_{ik}\}, \beta, \epsilon$, for any $0 \leq \varrho \leq \varrho_0$, we can either (i) provide $\{y_i\}, \{y_{i(k)}\}$ that satisfies LP20 or (ii) provide a (compressed) solution for LP5 such that $\sum_{(i,j)} w_{ij} x_{ij} \geq (1 - 3\epsilon/2)\beta$.*

Proof. Let $\gamma = \sum_{(i,j),k} w_k u^s_{ijk} - \sum_{i,k} w_k \varrho \zeta_{ik}$, $\quad \mathcal{B}(i) = \left\{ k \,\middle|\, \sum_j u^s_{ijk} - \varrho\zeta_{ik} > 0 \right\}$

$$\Delta(i,\ell) = \sum_{k \geq \ell, k \in \mathcal{B}(i)} w_\ell (\sum_j u^s_{ijk} - \varrho\zeta_{ik}) + \sum_{k < \ell, k \in \mathcal{B}(i)} w_k (\sum_j u^s_{ijk} - \varrho\zeta_{ik})$$

$$\mathcal{S} = \left\{ (i, k^*_i) \,\middle|\, k^*_i = \operatorname*{argmax}_\ell \Delta(i,\ell) > \frac{\gamma w_\ell}{\beta} \right\} \qquad \text{and} \qquad \Delta_{\mathcal{S}} = \sum_{(i,k^*_i) \in \mathcal{S}} \Delta(i, k^*_i)$$

If $\Delta_{\mathcal{S}} \geq \epsilon\gamma/4$ then for $(i, k_i^*) \in \mathcal{S}$ we set $y'_{i(k)} = \frac{\gamma}{\Delta_{\mathcal{S}}} w_k$ for $k < k_i^*; k \in \mathcal{B}(i)$ and $y'_{i(k)} = \frac{\gamma}{\Delta_{\mathcal{S}}} w_{k_i^*}$ for $k > k_i^*; k \in \mathcal{B}(i)$. Every other $y'_{i(k)} = 0$. Set $y'_i = \max_k y'_{i(k)}$. It follows that:

$$\sum_{i,k} y'_{i(k)} \left(\sum_j u^s_{ijk} - \varrho\zeta_{ik} \right) = \frac{\gamma}{\Delta_{\mathcal{S}}} \sum_{(i,k_i^*)\in\mathcal{S}} \Delta(i, k_i^*) = \frac{\gamma}{\Delta_{\mathcal{S}}} \Delta_{\mathcal{S}} = \gamma$$

$$= \sum_{(i,j),k} w_k u^s_{ijk} - \sum_{i,k} w_k \varrho\zeta_{ik}$$

$$\sum_i y'_i = \sum_{(i,k_i^*)\in\mathcal{S}} y'_i = \frac{\gamma}{\Delta_{\mathcal{S}}} \sum_{(i,k_i^*)\in\mathcal{S}} w_{k_i^*} \leq \frac{\gamma}{\Delta_{\mathcal{S}}} \sum_{(i,k_i^*)\in\mathcal{S}} \Delta(i, k_i^*) = \frac{\beta}{\Delta_{\mathcal{S}}} \Delta_{\mathcal{S}} = \beta$$

Therefore we satisfy the lemma in this case. Therefore we focus in the case $\Delta_{\mathcal{S}} \leq \epsilon\gamma/4$. In this case let k'_{ij} the level which is immediately above $\max\{k_i^*, k_j^*\}$.

Define $\hat{u}_{ijk} = u^s_{ijk}$ for $k \geq k'_{ij}, k \in \mathcal{B}(i)$ and 0 otherwise. Likewise set $\hat{\zeta}_{ik} = \zeta_{ik}$ for $k > k_i^*, k \in \mathcal{B}(i)$ and 0 otherwise. Observe that

$$\sum_{(i,j),k} w_k \hat{u}_{ijk} - \sum_{i,k} w_k \varrho\hat{\zeta}_{ik} \geq \gamma - \Delta_{\mathcal{S}} \geq \gamma(1 - \epsilon/4) \qquad \text{and}$$

$$\sum_j \hat{u}_{ijk} = \sum_{j, k \geq k'_{ij}, k\in\mathcal{B}(i)} u_{ijk} \leq \frac{\Delta(i, k_i^* + 1) + \sum_{k>k_i^*} \varrho w_{k_i^*} \hat{\zeta}_{ik}}{w_{k_i^*}}$$

$$\leq \frac{\gamma}{\beta} + \sum_{k>k_i^*} \varrho\hat{\zeta}_{ik} = \frac{\gamma}{\beta} + \sum_k \varrho\hat{\zeta}_{ik}$$

from the definition of $\mathcal{S}$. Now order the vertices in an arbitrary order. We process the vertices in this order — when we are at vertex i, if for some k we have $\sum_j \hat{u}_{ijk} \leq \varrho\hat{\zeta}_{ik}$ then we set $x_{ij} = 0$ (Note that we do not need to know if the edge exists). If we are at a k such that $\sum_j \hat{u}_{ijk} > \varrho\hat{\zeta}_{ik}$ then we calculate $a_{ik} = (\sum_j \hat{u}_{ijk} - \varrho\hat{\zeta}_{ik}) / \sum_j \hat{u}_{ijk}$. We set $x_{ij} = \frac{\beta}{\gamma} a_{ik} \hat{u}_{ijk}$ (the fact that $u^s_{ijk} > 0$ implies that we already have the edge

stored). Observe that

$$\frac{\gamma}{\beta}\sum_{(i,j)} w_{ij}x_{ij} \geq \sum_{(i,j),k} w_k \hat{u}_{ijk} - \sum_{i,k} w_k \varrho \hat{\zeta}_{ik} \geq \gamma(1-\epsilon/4)$$

$$\frac{\gamma}{\beta}\sum_{j} x_{ij} \leq \sum_{k}\left(\sum_{j} \hat{u}_{ijk} - \varrho\hat{\zeta}_{ik}\right) \leq \frac{\gamma}{\beta}$$

Therefore we have a feasible solution for LP5. □

Theorem 6.3.1 follows from Lemmas 6.3.3–6.3.7. Note that the size of the fractional solution produced in Lemma 6.3.7, i.e., the number of non-zero edges in x_{ij} is $\tilde{O}(n^{1+1/p})$. We apply the recent algorithm in [34] to obtain an integral solution in $\tilde{O}(n^{1+1/p})$ space and time.

6.3.2 Deferred Degree Counter

Definition 6.3.8 (The Deferred Degree Counter Problem)**.** Consider the problem: We are given a weighted graph $G = (V, E, u)$ but the weights are *not revealed* to us. Instead we are given $\{\varsigma_{ij}\}$ with the promise that $\varsigma_{ij}/\chi \leq u_{ij} \leq \varsigma_{ij}\chi$. We can have a single pass over the $m = |E|$ edges and have to produce a smaller summary data structure $\mathcal{D}$ (ξ-*deferred degree counter*) storing only some of the edges. Once the data structure is constructed, then the exact weights u_{ij} of only those edges stored in our structure $\mathcal{D}$ are revealed to us. We then have to output *a* ξ-*degree counter*, a graph $H = (V, E', u^s)$ such that $(1-\xi)\sum_j u_{ij} \leq \sum_j u^s_{ij} \leq (1+\xi)\sum_j u_{ij}$. Note that this is implemented in a single pass over the graph.

Suppose that we know u_{ij} exactly and we only want to sparsify the graph. Then,

we can simply sample $O(\frac{1}{\xi^2}\log n)$ edges per vertex and by Chernoff bound, we will estimate $\sum_j u_{ij}$ within $(1 \pm \xi)$ factor. (i,j) is sampled with probability $p_{ij} = \min\{1, O(\frac{1}{\xi^2}\log n) \cdot \frac{u_{ij}}{\sum_{j'} u_{ij'}}\}$. This is the scheme used for the sparsification. The remaining part is the uncertainty in u_{ij} value. As shown in Section 5.5, it is actually safe to oversample, i.e., the same guarantee holds when we sample (i,j) with probability $p'_{ij} \geq p_{ij}$. So we calculate the upperbound of p_{ij} and use it for the sampling probability. Summarizing, we obtain the following lemma:

Lemma 6.3.5. *Given bound ϱ and $\{\varsigma_{ij}\}$ factor and edge weights are in $[1, O(\text{poly}\, n)]$, we can generate a deferred degree counter of size $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$. We can construct a graph $H = (V, E', u^s)$ such that $(1-\xi)\sum_j u_{ij} \leq \sum_j u^s_{ij} \leq (1+\xi)\sum_j u_{ij}$ when the edge weights of the stored edges are revealed. The algorithm runs in $O(m)$ time and $O(\frac{\chi^2 n}{\xi^2}\log n)$ space.* [3] *Note $\chi = n^{1/(4p)}$.*

Proof. The first step is to partition edges depending on ς_{ij} $O(\log n)$ layers where ς_{ij} differ by powers of Λ, i.e, the range of ς_{ij} for layers are $[1, \Lambda)$, $[\Lambda, \Lambda^2)$, $[\Lambda^2, \Lambda^3)$, $\cdots$. If we estimates $\sum_{j:\Lambda^{k-1} \leq \varsigma_{ij} < \Lambda^k} u_{ij}$ for all k, we can add them to estimate $\sum_j u_{ij}$.

Let $L_k = \{(i,j) | \Lambda^{k-1} \leq \varsigma_{ij} < \Lambda^k\}$. We sample each $(i,j) \in L_k$ with probability $p_e = \min\{1, \frac{C\chi^2}{\xi^2}\log n\}$ for a sufficiently large constant C and let S_k be the set of the edges sampled from L_k. We estimate $U = \sum_{(i,j)\in L_k} u_{ij}$ with $Z = \frac{1}{p_e}\sum_{(i,j)\in S_k} u_{ij}$. Note that $U = Z$ (if $p_e = 1$) or $\frac{u_{ij}}{p_e} \leq \frac{\xi^2}{C\log n}$. In the latter case, $(1-\xi)U \leq Z \leq (1+\xi)U$

[3] The running time and the space bound is for the insertion-only model. For the dynamic model, the running time is $O(\frac{\chi^2 m}{\xi^2}\log n)$ and the space is $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$.

with high probability by Chernoff bound (see Lemma 5.2.2). $\square$

6.3.3 Initial Solutions

We now show how to get initial solution to the bipartite MWM problem. Recall LP13 and :

$$\begin{aligned} &\tilde{\beta} = \min \textstyle\sum_i y_i \\ &y_{i(k)} + y_{j(k)} \geq w_k && \forall (i,j) \in \text{ level } k \\ &y_i - y_{i(k)} \geq 0 && \forall i, k \\ &y_i, y_{i(k)} \geq 0 \end{aligned} \tag{LP13}$$

In fact we will consider LP21 from whose solution we can easily construct a feasible solution for LP13 of the same objective value:

$$\begin{aligned} &\tilde{\beta} = \min \textstyle\sum_i y_i \\ &y_i + y_j \geq w_{ij} \quad \forall (i,j) \\ &y_i \geq 0 \end{aligned} \tag{LP21}$$

Consider the following online algorithm: We maintain y_i and on a new edge with weight w_{ij} which violates $y_i + y_j \geq w_{ij}$ we increases y_i, y_j by $\Delta(y_i), \Delta(y_j)$ to satisfy the constraint while maintaining $\Delta(y_i) = \Delta(y_j)$.

Consider the optimum fractional solution (or any feasible fractional solution for that matter) of LP21 denoted by $\{y_i^*\}$. If $y_i + y_j \geq y_i^* + y_j^*$ we clearly will not be increasing y_i, y_j. Therefore any increase must be due to the fact that $y_i < y_i^*$ or $y_j < y_j^*$. In the case $y_i < y_i^*$ we charge i the total increase due to the edge (that

is, $2\Delta(y_i)$). Otherwise we charge j an amount of $2\Delta(y_j)$. This charging depends on $\{y_i^*\}$; but it immediate that the total charge to i is at most $2y_i^*$. Therefore we have a 2 approximation for LP21. Multiplying each y_i by 1/3 gives us $\beta_0 \leq \beta^* \leq \hat{\beta} \leq (1+\epsilon)\beta^* \leq 3(1+\epsilon)\beta_0$ and $\epsilon_0 = 2/3$.

6.4 Discussion

In this chapter, we presented two approaches to primal-dual algorithms and a variety of techniques related to the multiplicative weights update method through the example of the bipartite matching problems. The two approaches and the technique described in Section 6.1.2 are applicable to other frameworks, for example, the fractional packing framework given by Plotkin et al. [84]. However, the technique described in Section 6.1.3 only works with the multiplicative weights update method due to its reliance on the specific potential function. We also presented the dual–primal approach in which we start from the dual LP rather than the primal LP.

When using the primal-dual algorithms, there is another aspect that we have to carefully consider: *Which variant of the primal-dual algorithm do we use?* Although the primal-dual algorithms for LPs in Section 2.3 use the same intuition of increasing the weight (or importance) of violated constraint exponentially, they vary in details: for example, they use different ways of constructing dual candidates and different potential functions to bound the number of iterations. This gives us an interesting choice.

As we already mentioned, the technique in Section 6.1.3 only works for the multiplicative weights update method. The analysis for the fractional packing framework in [84] relies on the global potential function unlike the constraint-by-constraint analysis in [11] (see Section 2.3.1).[4] The consequence of the difference is that the number of passes in the semi-streaming model can be reduced to a constant using the multiplicative weights update method while the fractional packing framework suffers from the barrier of $(\log n)$ factor in the number of passes.

On the other hand, the fractional packing framework uses a simpler way of constructing a dual candidate. It only relies on the violation of the current primal candidate rather than the history of the primal witnesses. In addition, it uses a "smoother" equation in constructing a dual candidate. Recall the update rule of the multiplicative weights update method:

$$u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{\mathcal{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathcal{M}(i,\mathbf{x}^t) \geq 0 \\ u_i^t(1-\epsilon)^{-\mathcal{M}(i,\mathbf{x}^t)/\rho} & \text{if } \mathcal{M}(i,\mathbf{x}^t) < 0 \end{cases}$$

whose derivate is not continuous at $\mathcal{M}(i,\mathbf{x}^t) = 0$. Also recall that the dual variable in the fractional packing framework is defined as:

$$\mathbf{y}_i \leftarrow \frac{1}{\mathbf{b}_i} \exp(\kappa \mathbf{A}_i \mathbf{x}/\mathbf{b}_i)$$

whose derivate is continuous. In the next two chapters, we demonstrate this point through the non-bipartite b–Matching problem. We also demonstrate that we can

[4] We do not claim that it is impossible to apply the technique to the fractional packing framework. We are simply stating that it requires a different analysis.

use properties of a polytope to design an efficient algorithm for (exponentially) large LPs.

Chapter 7

Application I: b-Matching

Chapter Outline: In this chapter, we present algorithms for the (uncapacitated) b–Matching problems in general graphs. As in Chapter 6, we solve the problem through LPs. However, the LP for the matching problem in general graphs contains exponentially many constraints and therefore, requires a different way to compute a dual candidate efficiently. In this chapter, we show that only a small number of constraints actually matter to the oracle and present an algorithm to find such constraints efficiently. We also give an algorithm to round the fractional solution efficiently. Again, for the simplicity of analysis, we use the insertion-only model in this chapter.

7.1 The Standard LP Formulation and Results

In this chapter we provide algorithms for finding fractional as well as integral solutions for the weighted maximum b-matching problem in general graphs. We address the uncapacitated version of the problem, defined as follows:

Definition 7.1.1. [86, Chapter 31] In the **b–Matching** problem we are given a weighted (possibly non-bipartite) graph $G = (V, E, \{w_{ij}\}, \{b_i\})$ where w_{ij} is the weight of edge (i, j) and b_i is the capacity of the vertex i. Let $|V| = n$ and $|E| = m$. For simplicity we assume b_i, w_{ij} are integers in $[0, \operatorname{poly} n]$. We can select an edge (i, j) with multiplicity x_{ij} such that $\sum_{j:(i,j)\in E} x_{ij} \leq b_i$ for all vertices i and the goal is to maximize $\sum_{(i,j)\in E} w_{ij}x_{ij}$. Let $B = \sum_i b_i$, note that without loss of generality, $B \geq n$.

The standard exact linear programming formulation for b–Matching has m variables and 2^n constraints. We begin with the standard LP formulation.

Definition 7.1.2 (Volume of Sets & Odd-Sets)**.** Given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, and non-negative values b_i for each $i \in V$, define the **volume** of a set $U \subseteq V$ to be $||U||_b = \sum_{i\in U} b_i$. Define $\mathcal{O} = \{U \mid ||U||_b \text{ is odd }\}$ and $\mathcal{O}_\delta = \{U \mid U \in \mathcal{O}; ||U||_b \leq 1/\delta\}$.

Consider the (undirected) formulation LP22 (See also [86, Chapter 31]).

$$\begin{aligned}
\beta^* = \text{LP22}(\mathbf{b}) = \quad & \max \textstyle\sum_{(i,j)} w_{ij}x_{ij} \\
& \textstyle\sum_j x_{ij} \leq b_i && \forall i \in V \\
& \textstyle\sum_{(i,j):i,j\in U} x_{ij} \leq \lfloor ||U||_b/2 \rfloor && \forall U \in \mathcal{O} \\
& x_{ij} \geq 0
\end{aligned} \tag{LP22}$$

Then the variable x_{ij} (which is the same as x_{ji}) corresponds to the fractional relaxation of the "multiplicity" of the edge (i, j) in the uncapacitated b–Matching. If $(i, j) \notin E$ then $x_{ij} = 0$ throughout this paper. We assume $(i, i) \notin E$; no self loops are allowed. The constraints in the primal LP22 correspond to the vertices and odd sets. It is well known that the formulation LP22 has an integral optimum solution when b_i are integers.

Statement of Results and Roadmap: Theorems 7.1.3 and 7.1.4 summarize the results for b–Matching problem. The overall algorithm is presented in Sections 7.2 and 7.3 along with the important parts of the proof. The full details of the proof is modularized and are in Sections 7.6 and 7.7, which handle finding the initial solution and computing the updates efficiently respectively. To avoid repetitions, we discuss the capacitated problem alongside the uncapacitated problem while finding the initial solution (Section 7.6). Section 7.8 proves the rounding algorithm of Theorem 7.1.4 and is self contained.

Theorem 7.1.3 (Fractional b–Matching)**.** *For any $0 < \delta \leq 1/16$ we find a $(1 - 14\delta)$-approximate maximum fractional weighted b-Matching using additional "work" space (space excluding the read-only input) $O(n \operatorname{poly}\{\delta^{-1}, \ln n\})$ and making $R = O(\delta^{-4} \ln n)$ passes over the list of edges. The running time is $O(mT + n \operatorname{poly}\{\delta^{-1}, \ln n\})$ for $T = O(\delta^{-5} \ln n)$.*

Theorem 7.1.4 (Integral b–Matching)**.** *Given a fractional b-matching $\mathbf{x}$ where x_{ij}*

indicates the amount with which the edge (i, j) is included in the matching, we can find an integral b–Matching of weight at least $(1-2\delta)\sum_{(i,j)} w_{ij}x_{ij}$ in $O(m'\delta^{-3}\ln(1/\delta))$ time and $O(m'/\delta^2)$ space where $m' = |\{(i,j)|x_{ij} > 0\}|$.

7.2 Algorithm Overview

It is not difficult to see that if we only retain the constraints for odd sets U where $U \in \mathcal{O}_\delta$ then a fractional solution of the modified system; when multiplied by $(1-\delta)$; satisfies LP22. The relaxed formulation that captures $1-\delta$ approximate solutions still has $n^{1/\delta}$ constraints which is exponential in $1/\delta$. In order to solve such a system efficiently, the approaches in in the previous chapter alone are not sufficient. The frameworks for LPs, maintain an infeasible primal candidate and seek to improve it iteratively by maintaining (the multiplicative weights update method) or constructing (the fractional packing) a dual candidate that indicates an "improvement direction". The multiplicative weights update method in its standard form uses the sign of the violation to decide between increasing or decreasing the dual weights – but computing and maintaining the $n^{1/\delta}$ violations and variables in the dual candidate is not efficient. Instead, we use the fractional packing framework [84]. The fractional packing framework uses linear operations (no signs) and compute (rather than maintain) $n^{1/\delta}$ dual variables. Although the computation can be done in small space on demand, i.e. computing one dual variable at a time when it is required, the process is time-inefficient.

We begin our solution starting from the question: *Can we develop a technique that bypasses the evaluation of these exponentially many weights?* This question is reminiscent of the maximum violation approach in the context of Ellipsoid algorithms (see [56]) where a separation oracle is sufficient from the perspective of polynomial time solvability even in the presence of exponentially many constraints. The approach of Padberg-Rao [82] is one such approach in the context of matching. However all these proofs rely on exact optimality – or more specifically, on the structure of the optimum solution, and the polynomial time solvability which is guaranteed is insufficient to design near linear time algorithms or even algorithms with just a small number of iterations. This development of weak separation coincided with a fascinating development of combinatorial techniques such as Total Dual Integrality (TDI), laminarity[1] etc., (see Giles and Pulleyblank [53], Cook [28], Cunningham and Marsh [31], and also Schrijver [86]). However the entire discussion was focused on the optimum primal and dual solutions – in fact such relationships do not exist for arbitrary feasible primal or dual solutions. It is then natural to ask the follow up question – *If notions such as laminarity do not exist for arbitrary feasible primal or dual solutions – then can we modify the polytope to achieve such properties?* The answer to the last question is surprising – if we **perturb** the polytope slightly, then two interesting theorems can be proven:

Theorem 7.2.1 (Perturbation Theorem)**.** *For a graph G with n vertices and any*

[1] L is defined as laminar if for any two sets $U, U' \in L$, $U \cap U'$ is either U, U' or $\emptyset$.

non-negative edge weights $\hat{x}_{ij} = \hat{x}_{ji}$ *such that* $\hat{x}_{ii} = 0$ *and* $\sum_j \hat{x}_{ij} \leq b_i$ *for all* i*; and* $\delta \in (0, \frac{1}{16}]$*, define:*

$$\hat{\lambda}_U = \frac{\sum_{(i,j):i,j\in U} \hat{x}_{ij}}{\tilde{b}_U} \quad \textit{where} \quad \tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - \frac{\delta^2 ||U||_b^2}{4} \qquad \textit{and} \qquad \hat{\lambda} = \max_{U\in\mathcal{O}_\delta} \hat{\lambda}_U$$

If $\hat{\lambda} \geq 1 + 3\delta$*, the set* $L_1 = \{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^3; U \in \mathcal{O}_\delta\}$ *defines a laminar family. Moreover for any* $x \geq 2$ *we have* $|\{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^x; U \in \mathcal{O}_\delta\}| \leq n^3 + (n/\delta)^{1+\delta^{(x-3)/2}}$.

In other words, Theorem 7.2.1 states that *if we were to focus at constraints that are almost as violated as the maximum violated constraint of the perturbed polytope, then those constraints correspond to a laminar family for any plausible (such as non-negative, obeying the vertex constraints, etc.) primal solution candidate* $\{\hat{x}_{ij}\}$. This can be viewed as a strengthening of the laminarity properties which were observed at the *optimum Dual* solution in [31, 86]. Theorem 7.2.1 extends the characterization to all primal candidates (modulo adding the perturbations). The intuitive reason is simple — if we were to ignore the floor and ceil functions then for a fixed $\hat{\lambda}_U$, the function $\sum_{(i,j):i,j\in U} \hat{x}_{ij}$ is a concave function of $||U||_b$. As a result if two such U_1, U_2 intersect at a non-singleton odd set $U_3 \neq U_1, U_2$ (the union $U_4 \neq U_1, U_2$ is also an odd set) then $\max\{\hat{\lambda}_{U_3}, \hat{\lambda}_{U_4}\}$ will exceed $\min\{\hat{\lambda}_{U_1}, \hat{\lambda}_{U_2}\}$ by δ^3. Of course, the floor and ceil functions, singleton or even set intersections cannot be ignored and more details are required, but the idea behind the proof remains the same. Theorem 7.2.1 is a standalone combinatorial characterization and is proved in Section 7.4. Interestingly the theorem also hints at a simple proof of TDI for b–Matching along the lines of Schrijver's proof for regular matchings; this is discussed in Section 7.4.1. However

Theorem 7.2.1, does not (yet) give us an algorithm. This leads us to the next main theorem:

Theorem 7.2.2. *For a graph G with n vertices and $\{\hat{x}_{ij}\}$ and the definitions of $\{\hat{\lambda}_U\}$ exactly as in the statement of Theorem 7.2.1 and $\delta \in (0, \frac{1}{16}]$, if $\hat{\lambda} \geq 1 + 3\delta$ we can find the set $L_2 = \{U : \hat{\lambda}_U \geq \hat{\lambda} - \frac{\delta^3}{10}; U \in \mathcal{O}_\delta\}$ in $O(m' + n\,\mathrm{poly}\{\delta^{-1}, \log n\})$ time using $O(n\delta^{-5})$ space where $m' = |\{(i,j)|\hat{x}_{ij} > 0\}|$.*

The proof of Theorem 7.2.2 relies on the fact that $L_2 \subseteq L_1$ is a laminar family as proved in Theorem 7.2.1, these two theorems are intended to be used in tandem along with the observation that if $\hat{\lambda}$ is small then we are in an easy case. The proof of Theorem 7.2.2 is presented in Section 7.5. We provide a sketch of a proof and algorithmic ideas here. There are two hurdles to overcome in Theorem 7.2.2: (i) How do we even know $\hat{\lambda}$ *efficiently*, i.e., in near linear time? and (ii) How do we find *all* sets in L_2 efficiently? An inefficient answer to (i) follows from the minimum odd-cut approach of Padberg and Rao [82], because large $\hat{\lambda}_U$ implies a small cut. But that approach uses exact Gomory-Hu trees and the computation of such is not known to be in near linear time. As regards (ii) we show that using the specific way $\hat{\lambda}$ is being found, we can find all sets in L_2 simultaneously.

Define $L_1(\ell) = \{U|U \in L_1, ||U||_b = \ell\}$ and $L_2(\ell) = \{U|U \in L_2, ||U||_b = \ell\}$ for $\ell \in [3, 1/\delta]$. Note that $L_1(\ell) \supseteq L_2(\ell)$. Observe that it suffices to identify $L_2(\ell)$ for some fixed ℓ – we can repeat this for different ℓ. Now, if we can reduce the problem of finding $L_2(\ell)$ to some problem of finding low cuts in an unweighted graph then

there exists near linear time algorithms for finding a *representation* of all small cuts [57, 20] (the equivalent of Gomory-Hu trees for small cuts). Note, that the algorithm of [82] still guarantees finding one cut.

Our solution (see Section 7.2.2) is to construct an unweighted graph G_φ with $p_{ij} = \lfloor \varphi \hat{x}_{ij} \rfloor$ parallel edges between i and j where $\varphi = 50/\delta^4$. We can merge all pairs of nodes which have more than 2φ edges between them, and delete nodes with degree larger than $2\varphi/\delta$ giving us a bounded degree graph which can be stored in small space. Subsequently, we add a special node s and construct unweighted graphs $G_\varphi(\ell, \tilde{\lambda})$ with the following two properties:

Property 1. If $\tilde{\lambda} - \frac{\delta^3}{100} < \hat{\lambda} \leq \tilde{\lambda}$, then (i) all sets in $L_2(\ell)$ have a cut which is at most $\kappa(\ell)$ and (ii) all odd sets of $G_\varphi(\ell, \tilde{\lambda})$ which do not contain s and have cut at most $\kappa(\ell)$ belong to $L_1(\ell)$. Here $\kappa(\ell) = \lfloor \varphi\tilde{\lambda}(1 - \delta^2\ell^2/2) \rfloor + \frac{12\ell}{\delta} + 1 < 2\varphi$.

Property 2. We show in Lemma 7.5.1 that we can extend the algorithm in [82] to efficiently extract a collection $\mathcal{L}$ of *maximal* odd-sets in $G_\varphi(\ell, \tilde{\lambda})$, not containing s and cut at most $\kappa(\ell)$ – such that any such set which is not chosen must intersect with some set in the collection.

If we have a maximal collection $\mathcal{L}$ then $\mathcal{L} \subseteq L_1(\ell)$ by condition (ii) of Property 1. Due to Theorem 7.2.1, the intersection of two such sets $U_1, U_2 \in L_1(\ell)$ will be either empty or of size ℓ by laminarity – the latter implies $U_1 = U_2$. Therefore the sets in $L_1(\ell)$ are disjoint. Any $U \in L_2(\ell) - \mathcal{L}$ has a cut of at most $\kappa(\ell)$ using condition (i) of Property 1 and therefore must intersect with some set in $\mathcal{L}$. This is impossible

because $U \in L_2(\ell)$ implies $U \in L_1(\ell)$ and $\mathcal{L} \subseteq L_1(\ell)$ and we just argued that the sets in $L_1(\ell)$ are disjoint! Therefore no such U exists and $L_2(\ell) \subseteq \mathcal{L}$.

We now have a complete algorithm: we perform a binary search over the estimate $\tilde{\lambda} \in [1 + 3\delta, \frac{3}{2} + \delta^2]$, and we can decide if there exists a set $U \in L_2(\ell)$ in time $O(n \operatorname{poly}(\delta^{-1}, \log n))$ as we vary $\ell, \tilde{\lambda}$. This gives us $\tilde{\lambda}$. We now find the collections $\mathcal{L}$ for each ℓ and compute all $\hat{\lambda}_U$ exactly (either remembering the $\hat{x}_{ij}$ of the the edges stored in G_φ or by another pass over G). We can now return $\cup_\ell L_2(\ell)$. The complete proof of Theorem 7.2.2 is in Section 7.5.

7.3 $(1-\epsilon)$-Approximate Fractional b-Matching

In this section we prove Theorem 7.1.3 using a primal-dual algorithm. Algorithm 27 is the main algorithm of this chapter. We first show how the two main theorems 7.2.1 and 7.2.2 are used in this algorithm. Using the notation of Algorithm 27 we prove:

Lemma 7.3.1. *Fix $\delta \in (0, \frac{1}{16}]$. If $\lambda > 1 + 8\delta$ then we can find $L = \{U | \lambda_U \geq \lambda - \delta^3/10; U \in \mathcal{O}_\delta\}$ in $O(m + n \operatorname{poly}\{\delta^{-1}, \ln n\})$ time. Moreover L defines a laminar family. Finally, for any $x \geq 2$ we have $|\{U : \lambda_U \geq \lambda - \delta^x; U \in \mathcal{O}_\delta\}| \leq n^3 + (n/\delta)^{1+\delta^{(x-3)/2}}$.*

Proof. Let $\underline{\lambda} = \max\{1, \max_i (1-4\delta)\lambda_i\} = \max\{1, \max_i \sum_j x_{ij}/b_i\}$ and $\hat{x}_{ij} = x_{ij}/\underline{\lambda}$. Let $\hat{\lambda}_U = \sum_{i,j \in U} \hat{x}_{ij}/\tilde{b}_U$ and $\hat{\lambda} = \max_{U \in \mathcal{O}_\delta} \hat{\lambda}_U$. Observe that $\lambda_U = \underline{\lambda}\hat{\lambda}_U$ and if $\lambda > \max_i \lambda_i$ then $\lambda = \underline{\lambda}\hat{\lambda}$. Moreover we satisfy $\sum_j \hat{x}_{ij} \leq b_i$.

Algorithm 27 Near Linear Time Approximation Scheme for b–Matching (Part I)

1: Let $||U||_b = \sum_{i \in U} b_i$ and $\mathcal{O} = \{U \mid ||U||_b \text{ is odd }\}$. **Fix** $\delta \in (0, \frac{1}{16}]$. Let $\mathcal{O}_\delta = \{U \mid U \in \mathcal{O}; ||U||_b \leq 1/\delta\}$.

2: Initialize $\epsilon = 1/8$. Find an initial solution (See Section 7.6) with $\beta = \beta_0$ (and $\beta^* \leq \beta_0 \leq 6\beta^*$), $\lambda \leq \lambda_0 = 12$ such that LP23 holds. Note $x_{ij} = x_{ji}$ throughout.

$$\mathbf{A}: \begin{cases} \sum_j x_{ij} \leq \lambda \tilde{b}_i & \forall i \in V \quad \text{where } \tilde{b}_i = (1-4\delta)b_i \\ \sum_{(i,j):i,j \in U} x_{ij} \leq \lambda \tilde{b}_U & \forall U \in \mathcal{O}_\delta \quad \text{where } \tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - \frac{\delta^2 ||U||_b^2}{4} \end{cases}$$

$$\mathcal{P}[\beta]: \begin{cases} \sum_{(i,j)} w_{ij} x_{ij} \geq (1-\delta)\beta \\ \mathcal{P}: \begin{cases} \sum_j x_{ij} \leq 6b_i & \forall i \in V \\ x_{ij} \geq 0 \end{cases} \end{cases} \tag{LP23}$$

3: Let $\alpha = 50\delta^{-3} \ln n$. Define $\lambda_i = \sum_j x_{ij}/\tilde{b}_i$ and $\lambda_U = \sum_{(i,j):i,j \in U} x_{ij}/\tilde{b}_U$ and $\lambda = \{\max_i \lambda_i, \max_{U \in \mathcal{O}_\delta} \lambda_U\}$.

4: The algorithm proceeds in superphases which are further subdivided into phases. A new superphase starts when $\lambda \leq 1 + 8\epsilon$ (we will be decreasing ϵ gradually). A new phase starts either at the start of a superphase or when $\lambda \leq (1-8\delta)\lambda_t$ where λ_t is the value of λ at start of phase t. Note $\epsilon \geq \delta$.

Algorithm 27 Near Linear Time Approximation Scheme for b–Matching (Part II)

1: **while** $\epsilon > \delta$ **do**

2: **while** in **phase** t **do**

3: Compute λ exactly (for every iteration within the phase) and find a laminar collection of odd sets L such that $U \notin L \Rightarrow \lambda_U \leq \lambda - \delta^3/2$. (**Lemma 7.3.1** proves the validity of lines $6, 7$.)

4: If $\lambda < \max\{1 + 8\epsilon, (1 - 8\delta)\lambda_t\}$ then end **phase** t (goto line 10).

5: Set $x_i = exp(\alpha\lambda_i)/\tilde{b}_i$ for $i \in V$. For $U \in L$ set $z_U = exp(\alpha\lambda_U)/\tilde{b}_U$ else $z_U = 0$.

6: Let $\mathcal{L}(\mathbf{x}', \varrho) = \sum_{(i,j)} w_{ij}x'_{ij} - \varrho\left(\sum_{(i,j)} x'_{ij}(x_i + x_j + \sum_{U \in L; i,j \in U} z_U)\right)$ for $\varrho \geq 0$. Let $\gamma = \sum_i x_i\tilde{b}_i + \sum_{U \in \mathcal{O}_\delta} z_U\tilde{b}_U$. Consider:

$$\sum_{(i,j)} \tilde{x}_{ij}(x_i + x_j + \sum_{U \in L; i,j \in U} z_U) \leq \gamma \qquad \text{s.t. } \tilde{\mathbf{x}} \in \mathcal{P}[\beta] \tag{LP24}$$

$$\mathcal{L}(\mathbf{x}', \varrho) \geq \beta - \varrho\gamma \qquad \text{s.t. } \mathbf{x}' \in \mathcal{P} \tag{LP25}$$

7: Use $O(\frac{1}{\delta})$ (parallel) calls to LP25 to find a $\{\tilde{x}_{ij}\}$ feasible for LP24 (**Lemma 7.3.2**). If any of the substeps fail then set $\beta \leftarrow (1 - \delta)\beta$ and repeat this same step till a feasible solution is found.

8: Set $x_{ij} \leftarrow (1 - \sigma)x_{ij} + \sigma\tilde{x}_{ij}$ where $\sigma = \epsilon/(4\lambda_0) = \epsilon/(48\alpha)$.

9: **end while**

10: If $(\epsilon = \delta)$ output result (line 13).

11: If $(\lambda_t \leq 1 + 8\epsilon)$ then start new **superphase** (and phase) with $\epsilon \leftarrow \max\{2\epsilon/3, \delta\}$ else start **phase** $t + 1$.

12: **end while**

13: **Output:** $\{x^s_{ij} = \frac{x_{ij}}{1+8\delta}\}$. Note $\sum_{(i,j)} w_{ij}x^s_{ij} \geq (1 - 14\delta)\beta^*$ and $\{x^s_{ij}\}$ is feasible for LP22 (**Theorem 7.3.3**).

Suppose that $\hat{\lambda} \leq 1 + 3\delta$ and $\underline{\lambda} = 1$. Then for all U we have $\lambda_U = \underline{\lambda}\hat{\lambda}_U \leq \underline{\lambda}\hat{\lambda} \leq \underline{\lambda}(1 + 3\delta) < 1 + 8\delta$ and $\max_i \lambda_i \leq (1 - 4\delta) \leq 1 + 8\delta$ for $\delta \in (0, \frac{1}{16}]$. This contradicts the assumption that $\lambda > 1 + 8\delta$. Therefore, if $\hat{\lambda} \leq 1 + 3\delta$ then we must have $\underline{\lambda} > 1$. Now consider the vertex i which defined $\underline{\lambda}$; then

$$\lambda \geq \lambda_i = \frac{\underline{\lambda}}{1 - 4\delta} \geq (1 + 4\delta)\underline{\lambda} \geq (1 + 3\delta)\underline{\lambda} + \delta\underline{\lambda} > \hat{\lambda}\underline{\lambda} + \delta \tag{7.3.1}$$

which implies $\lambda - \delta \geq \lambda_U$ for every U. In this case $L = \emptyset$ and $|\{U : \lambda_U \geq \lambda - \delta^x; U \in \mathcal{O}_\delta\}| = 0$ for $x \geq 2$. Therefore the remaining case is $\hat{\lambda} > 1 + 3\delta$. But in this case Theorems 7.2.1 and 7.2.2 apply! This is because we satisfy $\sum_j \hat{x}_{ij} \leq b_i$. To find L, compute $\underline{\lambda}, \hat{x}_{ij}$ and run the algorithm in Theorem 7.2.2 and check if $\hat{\lambda} > 1 + 3\delta$ based on the sets returned. If the check is true then we can compute $\lambda = \{\underline{\lambda}\hat{\lambda}, \max_i \lambda_i\}$ and return the sets satisfying $\lambda_U \geq \lambda - \delta^3/10$. □

The following lemma is a slight rewording of Lemmas 6.2.1 and 6.2.2 and we prove it in Section 7.7.

Lemma 7.3.2. *If $(1-4\delta)\beta^* \geq \beta$ then (i) we always solve LP25 (and do not decrease β) and (ii) we can solve LP24 using $O(1/\delta)$ (parallel) invocations of LP25 (for different $\varrho \geq 0$) and using the convex combination of two solutions. The solution for LP25 uses $O(m)$ time, $O(n/\delta)$ space, a single pass over the edges and outputs a solution with $O(n)$ non-zero edges.*

Theorem 7.3.3. *Algorithm 27 produces a feasible fractional b–Matching of weight at least $(1-14\delta)\beta^*$ in $R = O(\delta^{-4} \ln n)$ passes and invokes LP25 at most $T = O(\delta^{-5} \ln n)$ times.*

Proof. The first observation is that based on Theorem 7.3.2 the algorithm never decreases β once $\beta \leq (1-4\delta)\beta^*$. Therefore the final value of β is at least $(1-\delta)(1-4\delta)\beta^*$. Observe that the entire algorithm can be analyzed at the final value of the β. Since the constraints $\mathcal{P}[\beta_1] \Rightarrow \mathcal{P}[\beta_2]$ for $\beta_1 \geq \beta_2$, we apply induction that any step for β_1 continues to be a legitimate step for β_2. In effect we are running the algorithm simultaneously for all β.

When Algorithm 27 stops, all the constraints $\mathbf{A}$ are violated by at most a factor of $1+8\delta$. Scaling the x_{ij} to $x''_{ij} = x_{ij}/(1+8\delta)$ we ensure that all constraints are satisfied. Note that $\sum_j x''_{ij} \leq (1-4\delta)b_i$. Therefore for any $U \notin \mathcal{O}_\delta$ we have

$$2\sum_{(i,j):i,j\in U} x''_{ij} \leq \sum_{i\in U}\sum_j x''_{ij} \leq \sum_{i\in U}(1-4\delta)b_i \leq (1-4\delta)||U||_b \leq ||U||_b - 1$$

therefore all constraints of LP22 are satisfied. We are guaranteed $\sum_{(i,j)} w_{ij}x''_{ij} \geq (1+8\delta)^{-1}(1-\delta)\beta$ (the polytope $\mathcal{P}$ has the $(1-\delta)$ approximation built in). When Algorithm 27 stops, $\sum_{(i,j)} w_{ij}x_{ij} \geq (1+8\delta)^{-1}(1-\delta)^2(1-4\delta)\beta^* \geq (1-14\delta)\beta^*$. The rest of the proof will be similar in spirit to [84], however the analysis is quite different materially. We analyze the number of rounds within phase t; when $\lambda > \max\{1+8\delta, 1+6\epsilon, (1-\epsilon)\lambda_t\}$ and ϵ remains unchanged.

Define $z'_U = e^{\lambda_U\alpha}/\tilde{b}_U$ for all $U \in \mathcal{O}_\delta$. Note $z_U = z'_U$ for $U \in L$ and 0 otherwise. Denote $\{y_i\}, \{z'_U\}$ by the vector $\mathbf{u}(\mathcal{O}_\delta)$ and denote $\{y_i\}, \{z_U\}$ by the vector $\mathbf{u}(L)$.

$\mathbf{u}(L)^T\mathbf{A}\mathbf{x} = \sum_i \lambda_i e^{\lambda_i\alpha} + \sum_{U\in L}\lambda_U e^{\lambda_U\alpha}$ and $\gamma = \sum_i \tilde{b}_i y_i + \sum_{U\in L} z_U\tilde{b}_U = \sum_i e^{\lambda_i\alpha} + \sum_{U\in L} e^{\lambda_U\alpha}$. Observe that $e^{\lambda\alpha} \leq \gamma$ since $\lambda = \max_i \lambda_i$ or $\lambda = \max_{U\in\mathcal{O}_\delta}\lambda_U$ for some $U \in L$. Finally $\gamma \leq 2ne^{\lambda\alpha}$ since L is laminar and therefore has at most n sets. Obviously,

$\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\mathbf{x} \geq \mathbf{u}(L)^T\mathbf{A}\mathbf{x}$.

Define $\Psi = \sum_i \tilde{b}_i y_i + \sum_{U\in\mathcal{O}_\delta} z'_U \tilde{b}_U = \sum_i e^{\lambda_i\alpha} + \sum_{U\in\mathcal{O}_\delta} e^{\lambda_U\alpha}$ and note $\gamma \leq \Psi$. Now,

(i) If $\lambda_U \leq (1-\delta^2)\lambda \leq \lambda-\delta^2$ then the corresponding $e^{\lambda_U\alpha} \leq e^{\lambda\alpha-\delta^2\alpha} \leq e^{\lambda\alpha}e^{-50\delta^{-1}\ln n}$ $= e^{\alpha_t\lambda}/n^{(50/\delta)}$. There are at most $n^{1/\delta}$ such sets and therefore $\sum_{U:\lambda_U\leq(1-\delta^2)\lambda} e^{\lambda_U\alpha}$ $\leq e^{\lambda\alpha}/n^{(49/\delta)}$.

(ii) Likewise (assuming $\delta^{(x-3)/2} \geq 2$) if $\lambda_U \leq (1-\delta^{x+\Delta(x)})\lambda \leq \lambda - \delta^{x+\Delta(x)}$ then the corresponding $e^{\lambda_U\alpha} \leq e^{\alpha\lambda}/n^{50\delta^{x+\Delta(x)}}$. Using Theorem 7.2.1 we know that there are at most $n^3 + (n/\delta)^{1+\delta^{(x-3)/2}} \leq n^{\delta^{(x-3)/2}+4}$ such sets. We can set $\Delta(x) = \frac{3-x}{2}$ and the total contribution of $\sum_{U:\lambda_U\leq\lambda-\delta^{x+\Delta(x)}} e^{\lambda_U\alpha} \leq e^{\lambda\alpha}/n^{49\delta^{(x-3)/2}-4} \leq e^{\lambda\alpha}/n^{98-4} \leq e^{\lambda\alpha}/n^{94}$.

(iii) We now geometrically divide the interval $(x,3]$ (for the analysis) and recurse on $(\frac{x+3}{2},3]$ till $\delta^{(x-3)/2} < 2$. At this point the number of remaining constraints is small since $n^3 + (n/\delta)^{1+\delta^{(x-3)/2}} \leq 2(n/\delta)^3$. We will reach the point within $2+\log\log(1/\delta)$ iterations. Now for the remaining $U \in \mathcal{O}_\delta$ if $U \notin L$ then we have $\lambda_U \leq \lambda - \frac{\delta^3}{10}$. Each such $e^{\lambda_U\alpha} \leq e^{\lambda\alpha}e^{-\delta^3\alpha/10} = e^{\lambda\alpha}e^{-5\ln n} = e^{\lambda\alpha}/n^5$. Summing up over such $2(n/\delta)^3$ sets the total contribution is still at most $2e^{\lambda\alpha}\delta^{-3}/n^2$.

Since $\frac{1}{n^{49/\delta}} + \frac{2+\log\log(1/\delta)}{n^{94}} + \frac{2\delta^{-3}}{n^2} \leq \frac{1}{n}$; we get:

$$\sum_{U:\lambda_U\notin L} e^{\lambda_U\alpha} \leq \frac{e^{\lambda\alpha}}{n} \leq \frac{\gamma}{n} \quad \Longrightarrow$$
$$\Psi = \gamma + \sum_{U\notin L; U\in\mathcal{O}_\delta} e^{\lambda_U\alpha} \leq \gamma\left(1+\frac{1}{n}\right) \leq 4ne^{\alpha\lambda} \tag{7.3.2}$$

Since $U \in L \Rightarrow \lambda_U \geq (1-\delta^3/10)\lambda$, $\sum_j y_{ij} = \lambda_i \tilde{b}_i, \sum_{i,j\in U} y_{ij} = \lambda_U \tilde{b}_U$ we have:

$$\begin{aligned}\mathbf{u}(L)^T\mathbf{A}\mathbf{x} &= \sum_i e^{\lambda_i\alpha}\lambda_i + \sum_{U\in L}\lambda_U e^{\lambda_U\alpha} \geq \sum_{i:\lambda_i\geq(1-\delta^3/10)\lambda} e^{\lambda_i\alpha}\lambda_i + \sum_{U\in L}\lambda_U e^{\lambda_U\alpha} \\ &= \lambda\left(1-\frac{\delta^3}{10}\right)\left(\gamma - \sum_{i:\lambda_i<(1-\delta^3/10)\lambda} e^{\lambda_i\alpha}\right)\end{aligned}$$

but $\sum_{i:\lambda_i<(1-\delta^3/10)\lambda} e^{\lambda_i\alpha}$ can again be bounded as γ/n exactly as in step (i)-(iii), because $\lambda > 1$ and there are only n terms. This implies $\mathbf{u}(L)^T\mathbf{A}\mathbf{x} \geq \lambda(1-\delta^3/10)(1-1/n)\gamma$. From Equation (7.3.2), with some simplification,

$$\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\mathbf{x} \geq \mathbf{u}(L)^T\mathbf{A}\mathbf{x} > (1+4\epsilon)\Psi \tag{7.3.3}$$

Now for any $\tilde{\mathbf{x}} \in \mathcal{P}$, i.e., $\sum_j \tilde{x}_{ij} \leq 6b_i$, we have $\sum_{i,j\in U}\tilde{x}_{ij} \leq 6||U||_b/2$ as well as $\sum_j \tilde{x}_{ij} \leq 12\tilde{b}_i$ since $\delta \leq \frac{1}{8}$. $\sum_{i,j\in U}\tilde{x}_{ij} \leq 3||U||_b$ implies $\sum_{i,j\in U}\tilde{x}_{ij} \leq 12\tilde{b}_U$, since $\tilde{b}_U \geq \left\lfloor\frac{||U||_b}{2}\right\rfloor - \frac{1}{4}$ for $U \in \mathcal{O}_\delta$ and $||U||_b/\left(\left\lfloor\frac{||U||_b}{2}\right\rfloor - \frac{1}{4}\right)$ is maximized at $||U||_b = 3$. As a consequence $\tilde{\lambda}_i, \tilde{\lambda}_U \leq \lambda_0$. Since we repeatedly take convex combination of the current candidate solution $\mathbf{x}$ with a $\tilde{\mathbf{x}} \in \mathcal{P}$, and the initial solution satisfies $\lambda \leq \lambda_0$; we have λ_i, λ_U upper bounded by λ_0 throughout the algorithm. Therefore $\tilde{\lambda}_U = \sum_{i,j\in U}\tilde{x}_{ij}/\tilde{b}_U \leq 12$.

Note $\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\tilde{\mathbf{x}} = \sum_i \tilde{\lambda}_i e^{\lambda_i\alpha} + \sum_{U\in\mathcal{O}_\delta}\tilde{\lambda}_U e^{\lambda_U\alpha}$. $\mathbf{u}(L)^T\mathbf{A}\tilde{\mathbf{x}}$ has the second summand restricted to $\sum_{U\in L}$. Using first part of Equation (7.3.2):

$$\mathbf{u}(L)^T\mathbf{A}\tilde{\mathbf{x}} \geq \mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\tilde{\mathbf{x}} - \sum_{U\notin L}\tilde{\lambda}_U e^{\alpha\lambda_U} \geq \mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\tilde{\mathbf{x}} - \frac{12}{n}\gamma \tag{7.3.4}$$

After the update, let the new current solution be denoted by $\{x''_{ij}\}$. Let $\lambda''_i = \sum_j x''_{ij}/\tilde{b}_i; \lambda''_U = \sum_{i,j\in U} x''_{ij}/\tilde{b}_U$. Note $\lambda''_i = (1-\sigma)\lambda_i + \sigma\tilde{\lambda}_i$ and $\lambda_U = (1-\sigma)\lambda_U + \sigma\tilde{\lambda}_U$. Since $\lambda_i, \lambda_U \leq 12$ we have all $|\alpha\sigma(\tilde{\lambda}_i - \lambda_i)|$ and $|\alpha\sigma(\tilde{\lambda}_U - \lambda_U)| \leq \epsilon/4$. For $|\Delta| \leq \frac{\epsilon}{4} \leq \frac{1}{4}$;

we have $e^{a+\Delta} \leq e^a(1+\Delta+\epsilon|\Delta|/2)$. Therefore:

$$e^{\alpha\lambda_i''} \leq e^{\alpha\lambda_i}\left(1+\sigma\alpha(\tilde{\lambda}_i-\lambda_i)+\epsilon\sigma\alpha(\tilde{\lambda}_i+\lambda_i)\right) \quad \text{and}$$

$$e^{\alpha\lambda_U''} \leq e^{\alpha\lambda_U}\left(1+\sigma\alpha(\tilde{\lambda}_U-\lambda_U)+\epsilon\sigma\alpha(\tilde{\lambda}_U+\lambda_U)\right)$$

Rearranging and summing over i, U we get

$$\begin{aligned}
\Psi'' &= \sum_i e^{\alpha\lambda_i''} + \sum_{U\in\mathcal{O}_\delta} e^{\alpha\lambda_U''} \\
&\leq \Psi + \left(1+\frac{\epsilon}{2}\right)\sigma\alpha\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\tilde{\mathbf{x}} - \left(1-\frac{\epsilon}{2}\right)\sigma_t\alpha_t\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\mathbf{x} \\
&\leq \Psi + \sigma\alpha\left(1+\frac{\epsilon}{2}\right)\left(\mathbf{u}(L)^T\mathbf{A}\tilde{\mathbf{x}}+\frac{12}{n}\gamma\right) - \sigma\alpha\left(1-\frac{\epsilon}{2}\right)\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\mathbf{x} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Using Eqn. 7.3.4)} \\
&\leq \Psi + \sigma\alpha\left(1+\frac{\epsilon}{2}\right)\left(1+\frac{12}{n}\right)\gamma - \sigma\alpha\left(1-\frac{\epsilon}{2}\right)\mathbf{u}(\mathcal{O}_\delta)^T\mathbf{A}\mathbf{x} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Using LP24, } \mathbf{u}(L)^T\mathbf{A}\tilde{\mathbf{x}} \leq \gamma) \\
&\leq \Psi + \sigma\alpha\left(1+\frac{\epsilon}{2}\right)\left(1+\frac{12}{n}\right)\gamma - \sigma\alpha\left(1-\frac{\epsilon}{2}\right)(1+4\epsilon)\Psi \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Using Equation 7.3.3)} \\
&\leq (1-\epsilon\sigma\alpha)\Psi \qquad \text{(Using } \gamma \leq \Psi \text{ and } \tfrac{1}{n} \ll \delta \leq \epsilon \leq \tfrac{1}{6})
\end{aligned}$$

Therefore the potential decreases. Note that between two phases $e^{(1-8\delta)\lambda_t\alpha} \leq \Psi \leq 4ne^{\lambda_t\alpha}$ from Equation (7.3.2). This implies that each phase will end within $O(\frac{\ln n}{\epsilon\sigma\alpha} + \frac{\delta\lambda_t\alpha}{\epsilon\sigma\alpha})$ updates to $\mathbf{x}$; which is $O(\frac{\lambda_0 \ln n}{\epsilon^2} + \frac{\lambda_0\lambda_t\delta^{-2}\ln n}{\epsilon^2})$ which is $O(\frac{\delta^{-2}\ln n}{\epsilon^2})$ since $1+8\epsilon \leq \lambda_t \leq 1+12\epsilon$ (the previous value of ϵ being larger by a factor 3/2 and $\lambda_0 = O(1)$) for the superphase containing the phase t. Therefore the number of updates in a phase is bounded by $O(\frac{\delta^{-2}\ln n}{\epsilon^2})$. The number of phases in a superphase is at most $\ln_{\frac{1}{1-8\delta}} \frac{1+12\epsilon}{1+8\epsilon} = O(\frac{\epsilon}{\delta})$. Therefore the number of updates in a superphase is $O(\frac{\delta^{-3}\ln n}{\epsilon})$.

Now ϵ is decreased by a factor of $3/2$ and therefore the last two terms dominate (corresponding to $\epsilon = \delta$ and $\epsilon \in [\delta, 3\delta/2]$, giving us $R = O(\delta^{-4} \ln n)$. Note each iteration uses one pass and invokes LP25 $O(\frac{1}{\delta})$ times in parallel. This proves the bound on T once we have $\beta^*(1-4\delta) \geq \beta$. The number of extra steps in decreasing β is at most $O(\frac{1}{\delta})$ since $6\beta^* \geq \beta_0 \geq \beta^*$ and is absorbed in $O(\delta^{-4} \ln n)$. $\square$

Based on Theorem 7.3.3, we can conclude Theorem 7.1.3.

7.4 Perturbation Theorem and Combinatorial Characterizations

We observe a simple fact before we prove Theorem 7.2.1.

Fact 7.4.1. Recall $\tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - f(||U||_b)$ where $f(\ell) = \frac{\delta^2 \ell^2}{4}$ and $\delta \in (0, \frac{1}{16}]$. We can verify that $f(\ell)$ is convex, monotonic for $0 \leq \ell \leq 2/\delta$ and:

($\mathcal{F}1$): For $3 \leq ||U||_b \leq 2/\delta$ (irrespective of odd or even) we have $\tilde{b}_U \geq (1-\delta) \left\lfloor \frac{||U||_b}{2} \right\rfloor$.

($\mathcal{F}2$): For any ℓ_1, ℓ_2; $f(\ell_1) + f(\ell_2) = f(\ell_1 + \ell_2 - 1) - (2\ell_1\ell_2 - 2\ell_1 - 2\ell_2 + 1)\frac{\delta^2}{4}$.

($\mathcal{F}3$): For integers $\ell_1, \ell_2, \ell_3, \ell_4 \in [3, 2/\delta]$ such that $\ell_1 + 2t \leq \ell_2 \leq \ell_3 \leq \ell_4 - 2t$ and $\ell_1 + \ell_4 = \ell_2 + \ell_3$, $f(\ell_2) + f(\ell_3) \leq f(\ell_1) + f(\ell_4) - 2t^2\delta^2$.

Theorem 7.2.1. *For a graph G with n vertices and any non-negative edge weights $\hat{x}_{ij} = \hat{x}_{ji}$ such that $\hat{x}_{ii} = 0$ and $\sum_j \hat{x}_{ij} \leq b_i$ for all i; and $\delta \in (0, \frac{1}{16}]$, define:*

$$\hat{\lambda}_U = \frac{\sum_{(i,j):i,j \in U} \hat{x}_{ij}}{\tilde{b}_U} \quad \textit{where} \quad \tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - \frac{\delta^2 ||U||_b^2}{4} \qquad \textit{and} \qquad \hat{\lambda} = \max_{U \in \mathcal{O}_\delta} \hat{\lambda}_U$$

If $\hat{\lambda} \geq 1 + 3\delta$, the set $L_1 = \{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^3; U \in \mathcal{O}_\delta\}$ defines a laminar family. Moreover for any $x \geq 2$ we have $|\{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^x; U \in \mathcal{O}_\delta\}| \leq n^3 + (n/\delta)^{1+\delta^{(x-3)/2}}$.

Proof. Consider two sets $A_1, A_2 \in \mathcal{O}_\delta$ such that $\hat{\lambda}_{A_1}, \hat{\lambda}_{A_2} \geq \hat{\lambda} - \delta^x > 1 + 2\delta$ (since $x \geq 2$) and neither $A_1 - A_2, A_2 - A_1 \neq \emptyset$. For any set U (with $||U||_b \geq 1$, even or odd, large or small) define $\hat{x}_U = \sum_{(i,j):i,j \in U} \hat{x}_{ij}$ and $\tilde{b}_U$. For $||U||_b = 1$ we have $\hat{x}_U = 0$. Let $\hat{\lambda}_U = \hat{x}_U / \tilde{b}_U$. There are now two cases.

Case I: $||A_1 \cap A_2||_b$ is even. Let $||A_1 \cap A_2||_b = ||D||_b = 2t$.

Let $Q_1 = \sum_{i \in D} \sum_{j \in A_1 - A_2} \hat{x}_{ij}$ (the cut between D and $A_1 - A_2$ using the edge weights $\hat{x}_{ij}$) and $Q_2 = \sum_{i \in D} \sum_{j \in A_2 - A_1} \hat{x}_{ij}$. Without loss of generality, assume that $Q_1 \leq Q_2$ (otherwise we can switch A_1, A_2). Let $C = A_1 - A_2$ and $A = A_1$. Let $2\ell - 1 = ||C||_b$ which is odd. Then, using $Q_1 \leq Q_2$ and definitions of $\hat{x}_C, \hat{x}_D$ we have $\hat{x}_C = \hat{x}_A - Q_1 - \hat{x}_D$ and $\hat{x}_D \leq \frac{1}{2}(\sum_{i \in D} \sum_j \hat{x}_{ij} - Q_1 - Q_2) \leq \frac{||D||_b}{2} - \frac{Q_1 + Q_2}{2}$. This implies:

$$\hat{x}_C \geq \hat{x}_A - \frac{||D||_b}{2} - \frac{Q_1}{2} + \frac{Q_2}{2} \geq \hat{x}_A - \frac{||D||_b}{2} = \hat{x}_A - t \tag{7.4.1}$$

Now $\hat{x}_A = \hat{\lambda}_A \tilde{b}_A > (1 + 3\delta)(1 - \delta) \left\lfloor \frac{||A||_b}{2} \right\rfloor \geq \left\lfloor \frac{||A||_b}{2} \right\rfloor$ using Condition $\mathcal{F}1$, Fact 7.4.1 for $\delta \leq \frac{1}{8}$, and the lower bound on $\hat{\lambda}$. Therefore $\hat{x}_A > t$ and $\hat{x}_C > 0$ which means $||C||_b \geq 3$. Therefore we can refer to $\tilde{b}_C, \hat{\lambda}_C$. Since $||D||_b = ||A||_b - ||C||_b$,

$$\begin{aligned} \tilde{b}_A - \tilde{b}_C &= \left\lfloor \frac{||A||_b}{2} \right\rfloor - f(||A||_b) - \left\lfloor \frac{||C||_b}{2} \right\rfloor + f(||C||_b) \\ &= \frac{||D||_b}{2} - (f(||A||_b) - f(||C||_b)) = t - t\delta(t + 2\ell - 1)\delta \geq (1 - \delta)t \end{aligned} \tag{7.4.2}$$

where the last line uses $\frac{1}{\delta} \geq ||A||_b \geq (2t + 2\ell - 1)$ because $A \in \mathcal{O}_\delta$. From Equa-

tions (7.4.1) and (7.4.2), and $\hat{x}_C = \hat{\lambda}_C \tilde{b}_C, \hat{x}_A = \hat{\lambda}_A \tilde{b}_A$ we get:

$$\hat{\lambda}\tilde{b}_C \geq \hat{\lambda}_C \tilde{b}_C = \hat{x}_C \geq \hat{x}_A - t = \hat{\lambda}_A \tilde{b}_A - t \geq (\hat{\lambda} - \delta^x)\tilde{b}_A - t = \hat{\lambda}\tilde{b}_A - \delta^x \tilde{b}_A - t$$

$$\geq \hat{\lambda}(\tilde{b}_C + (1-\delta)t) - \delta^x \tilde{b}_A - t > \hat{\lambda}\tilde{b}_C + (1+3\delta)(1-\delta)t - \delta^x \tilde{b}_A - t \geq \hat{\lambda}\tilde{b}_C + \delta t - \delta^x \tilde{b}_A$$

Since $\tilde{b}_A \leq 1/\delta$ this implies that $t < \delta^{x-1}\tilde{b}_A \leq \delta^{x-2}$ which contradicts $A_1 \cap A_2 \neq \emptyset$ for $x \geq 2$.

Case II: $||A_1 \cap A_2||_b$ is odd. Let $C = A_1 \cup A_2$, and $D = A_1 \cap A_2$. Let $||A_1||_b = \ell_1 ||A_2||_b = \ell_2$. Even if $||C||_b \geq 1/\delta$ extend the definitions $\tilde{b}_C = \left\lfloor \frac{||C||_b}{2} \right\rfloor - f(||C||_b)$ and $\hat{\lambda}_C = \hat{x}_C / \tilde{b}_C$. Now $\hat{x}_A \leq \frac{||C||_b}{2}$ since $\sum_j \hat{x}_{ij} \leq b_i$. Note that if $||C||_b \geq 1/\delta$ then using Condition $\mathcal{F}1$, Fact 7.4.1:

$$\tilde{b}_C \geq (1-\delta)\left\lfloor \frac{||C||_b}{2} \right\rfloor = (1-\delta)\frac{||C||_b}{2}\left(1 - \frac{1}{||C||_b}\right) \geq (1-\delta)^2 \frac{||C||_b}{2}$$

which implies that $\hat{\lambda}_C \leq (1-\delta)^{-2} \leq 1 + 3\delta < \hat{\lambda}$. Now, we always have: $\hat{x}_C + \hat{x}_D = \hat{x}_{A_1} + \hat{x}_{A_2}$ and $\left\lfloor \frac{||C||_b}{2} \right\rfloor + \left\lfloor \frac{||D||_b}{2} \right\rfloor = \left\lfloor \frac{||A_1||_b}{2} \right\rfloor + \left\lfloor \frac{||A_2||_b}{2} \right\rfloor$. Therefore:

$$\hat{x}_C + \hat{x}_D = \hat{x}_{A_1} + \hat{x}_{A_2} = \hat{\lambda}_{A_1}\tilde{b}_{A_1} + \hat{\lambda}_{A_2}\tilde{b}_{A_2} \geq (\hat{\lambda} - \delta^x)(\tilde{b}_{A_1} + \tilde{b}_{A_2}) \tag{7.4.3}$$

If $||D||_b = 1$, then by Condition $\mathcal{F}3$ in Fact 7.4.1: $\tilde{b}_C = \tilde{b}_{A_1} + \tilde{b}_{A_2} - \frac{\delta^2}{4}(2\ell_1\ell_2 - 2\ell_1 - 2\ell_2 + 1)$:

$$\begin{aligned}\hat{\lambda}\tilde{b}_C \geq \hat{\lambda}_C \tilde{b}_C = \hat{x}_C &\geq (\hat{\lambda} - \delta^x)(\tilde{b}_{A_1} + \tilde{b}_{A_2}) \geq \hat{\lambda}(\tilde{b}_{A_1} + \tilde{b}_{A_2}) - \delta^x(\tilde{b}_{A_1} + \tilde{b}_{A_2}) \\ &\geq \hat{\lambda}\tilde{b}_C + \frac{\delta^2\hat{\lambda}}{4}(2\ell_1\ell_2 - 2\ell_1 - 2\ell_2 + 1) - \delta^x\left(\frac{\ell_1 + \ell_2 - 2}{2}\right)\end{aligned}$$

since $\tilde{b}_{A_1} + \tilde{b}_{A_2} \leq (\ell_1 + \ell_2 - 2)/2$. Therefore we would have a contradiction if

$$\hat{\lambda}(2\ell_1\ell_2 - 2\ell_1 - 2\ell_2 + 1) - 2\delta^{x-2}(\ell_1 + \ell_2 - 2) > 0 \tag{7.4.4}$$

Observe that for $x \geq 3$ the term $2\delta^{x-2}(\ell_1 + \ell_2 - 2)$ is at most 2 whereas $(2\ell_1\ell_2 - 2\ell_1 - 2\ell_2 + 1) \geq 7$ since $3 \leq \ell_1, \ell_2 \leq \frac{1}{\delta}$. Since $\hat{\lambda} > 1$ we have a contradiction for $||D||_b = 1, x \geq 3$.

Now consider $||D||_b \geq 3$. Without loss of generality, $||A_2 - D||_b \geq ||A_1 - D||_b$. Let $||A_1 - D||_b = 2t$. Using Condition $\mathcal{F}3$ in Fact 7.4.1, $\tilde{b}_C + \tilde{b}_D \leq \tilde{b}_{A_1} + \tilde{b}_{A_2} - 2t^2\delta^2$. Note $\hat{\lambda}_D \leq \hat{\lambda}$. From Equation (7.4.3):

$$\begin{aligned}\hat{\lambda}\left(\tilde{b}_C + \tilde{b}_D\right) &\geq \hat{\lambda}_C\tilde{b}_C + \hat{\lambda}_D\tilde{b}_D = \hat{x}_C + \hat{x}_D \geq \hat{\lambda}(\tilde{b}_{A_1} + \tilde{b}_{A_2}) - \delta^x(\tilde{b}_{A_1} + \tilde{b}_{A_2}) \\ &\geq \hat{\lambda}(\tilde{b}_C + \tilde{b}_D) + 2t^2\delta^2\hat{\lambda} - \delta^x(\tilde{b}_{A_1} + \tilde{b}_{A_2}) \qquad (7.4.5)\end{aligned}$$

Again, this is infeasible if $x \geq 3$ since $\tilde{b}_{A_1} + \tilde{b}_{A_2} \leq 2/\delta$ and $\hat{\lambda} \geq 1$. Therefore for $x \geq 3$, in all cases we arrived at a contradiction to $A_1 \cap A_2 \neq \emptyset$. Thus we have proved that $\{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^3; U \in \mathcal{O}_\delta\}$ is a laminar family. We now prove the second part.

Consider $L'_\ell = \{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^x; U \in \mathcal{O}_\delta; ||U||_b = \ell\}$. From **Case I**, no two distinct sets $A_1, A_2 \in L'_\ell$ intersect when $||A_1 \cap A_2||_b$ is even. From **Case II** for $\ell \geq 5$, they cannot have $||D||_b = 1$ because $(2\ell^2 - 4\ell + 1) - 2(2\ell - 2) > 0$ for $\ell \geq 5$. Note $||A_1 - D||_b = ||A_2 - D||_b$ because $||A_1||_b = ||A_2||_b = \ell$. Moreover for $t \geq \delta^{(x-3)/2}$ we would have $2t^2\delta^2\hat{\lambda} > \delta^x(\tilde{b}_{A_1} + \tilde{b}_{A_2})$ in Equation 7.4.5. Therefore two distinct sets $A_1, A_2 \in L'_\ell$ which intersect, cannot differ by more than $\delta^{(x-3)/2}$ elements. This means that $|L'_\ell| \leq n\,(n/\delta)^{\delta^{(x-3)/2}}$ for $\ell \geq 5$ — to see this choose a maximal collection of disjoint sets and every other set has to intersect one of these sets. If we fix a set we can throw out $\delta^{(x-3)/2}$ elements in $\ell^{\delta^{(x-3)/2}}$ ways and include new elements in $n^{\delta^{(x-3)/2}}$ ways. Note $|L'_3| \leq n^3$ and $\ell \leq 1/\delta$. Thus the total number of sets is

$n^3 + (n/\delta)(n/\delta)^{\delta^{(x-3)/2}}$. The lemma follows. $\square$

7.4.1 A Simple Proof of TDI for the b–Matching Polytope

The idea of a laminar family in the context of maximum matching was first explored by Cunningham and Marsh [31]. They showed that there exists an integral optimum dual solution for the maximum matching problem; that is, LP22 is total dual integral (TDI) for $b_i = 1$. Schrijver [86, pages 441-442, vol A], gave an alternate and simpler proof. We show in Theorem 7.4.2 that the claim extends to b–Matching. However we separate the proof of laminarity from the proof of integrality. In this paper we only need laminarity — moreover the perturbed capacities are non-integral. The proof of Theorem 7.4.2 indicates a three-step choice and the functional forms are reminiscent of the perturbation in Algorithm 27 (but other convex functions with large second derivatives work for both the algorithm and Theorem 7.4.2). Intuitively we extend the statement that "the optimum dual solution is laminar" to the statement that *the most violated constraints in the neighborhood of any infeasible primal define a laminar family.* The optimum primal solution is of course a boundary point of infeasible primal solutions. Without further ado we present Theorem 7.4.2.

Theorem 7.4.2. *Consider the dual of LP22, presented in LP26*

$$\begin{aligned} &\min \sum_i b_i y_i + \sum_{U \in \mathcal{O}} \lfloor ||U||_b/2 \rfloor \, z_U \\ &y_i + y_j + \sum_{U \in \mathcal{O}; i,j \in U} z_U \geq w_{ij} \quad \forall (i,j) \in E \\ &y_i, z_U \geq 0 \qquad\qquad\qquad\qquad \forall i \in V; \forall U \in \mathcal{O} \end{aligned} \tag{LP26}$$

There exists an optimal solution of LP26 such that $L = \{U : z_U \neq 0\}$ is a laminar[2] *family. Recall, $\mathcal{O} = \{U | U \subseteq V; ||U||_b$ is odd $\}$ where $||U||_b = \sum_{i \in U} b_i$ and b_i are integers. If w_{ij} are integral it also follows that there exists an optimum solution LP26 which is integral as well as laminar.*

Proof. We follow the proof in [86], page 441-442. Let $\mathfrak{S}$ be the set of optimal solutions of LP26. Let $\mathfrak{S}_2 \subseteq \mathfrak{S}$ be the subset of optimal solutions which **minimize** $\boldsymbol{\sum_{z_U \neq 0} z_U ||U||_b}$ among the optimum solutions. We choose a solution from $\mathfrak{S}_2$ that **maximizes** $\boldsymbol{\sum_{z_U \neq 0} z_U ||U||_b^2}$. This is a **three** step choice. We show that L is a laminar family for the optimal solution we have chosen.

Suppose that L is not a laminar family. Then, there exist $A, B \in L$ such that (a) $z_A, z_B \neq 0$ and (b) $A \cap B \neq \emptyset$, A or B. There are two cases: $||A \cap B||_b$ is either even or odd. In both cases, we change the solution (while preserving the objective value and the feasibility of the solution).

1. $||A \cap B||_b$ **is even:** Let $z = \min\{z_A, z_B\}$. We reduce z_A and z_B by z and increase z_{A-B}, z_{B-A} by z. Observe that both $A - B, B - A$ are nonempty and odd. We now increase every y_i for $i \in A \cap B$ by z. These changes preserve the feasibility and the objective value of the solution. On the other hand, they decrease $\sum_{z_U \neq 0} z_U ||U||_b$ because we replace A and B by $A - B$ and $B - A$ (obviously, $||A - B||_b < ||A||_b$ and $||B - A||_b < ||B||_b$). Therefore, it contradicts the fact that the chosen solution belongs to $\mathfrak{S}_2$.

[2] $\mathcal{U}$ is laminar if for any two sets $U_1, U_2 \in \mathcal{U}$ we have $U_1 \bigcap U_2$ to be U_1 or U_2 or $\emptyset$.

2. $||A \cap B||_b$ **is odd:** Let $z = \min\{z_A, z_B\}$. We reduce z_A and z_B by z and increase $z_{A\cup B}$, $z_{A\cap B}$ by z. Again, these changes preserve the feasibility and the objective value of the solution. Since $||A \cup B||_b + ||A \cap B||_b = ||A||_b + ||B||_b$ and $||A \cup B||_b > ||A||_b, ||B||_b$, $||A \cup B||_b^2 + ||A \cap B||_b^2 \geq ||A||_b^2 + ||B||_b^2$. So $\sum_{z_U \neq 0} z_U ||U||_b^2$ increases which contradicts the fact that the solution maximizes $\sum_{z_U \neq 0} z_U ||U||_b^2$.

Therefore, L is a laminar family. For the second part of the theorem, use Corollary 31.3a [86, page 553, vol A] which shows that LP22, dual of LP26, is total dual integral (TDI). Apply the transformations above starting from that integral optimum solution. Observe that the parameter z is integral and the transformations preserve integrality. □

7.5 Finding a Laminar Family of Dense Odd Sets

In this section, we prove Theorem 7.2.2. We first state Lemma 7.5.1 which would be used in the proof of Theorem 7.2.2. We then state and prove the theorem and finish the section with the proof of Lemma 7.5.1.

Lemma 7.5.1. *Given an unweighted graph G with parameters κ, ϕ and a special node s, in time $O(n \operatorname{poly}(\delta^{-1}, \log n))$ we can identify a collection $\mathcal{L}$ of odd-sets which (i) do not contain s (ii) define cut of at most κ in G and (iii) every other odd set not containing s and with a cut less than κ intersects with a set in $\mathcal{L}$.*

Theorem 7.2.2. *For a graph G with n vertices and any non-negative edge weights $\hat{x}_{ij} = \hat{x}_{ji}$ such that $\hat{x}_{ii} = 0$ and $\sum_j \hat{x}_{ij} \leq b_i$ for all i; and $\delta \in (0, \frac{1}{16}]$, define:*

$$\hat{\lambda}_U = \frac{\sum_{(i,j):i,j\in U} \hat{x}_{ij}}{\tilde{b}_U} \quad \text{where} \quad \tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - \frac{\delta^2 ||U||_b^2}{4} \qquad \text{and} \qquad \hat{\lambda} = \max_{U\in\mathcal{O}_\delta} \hat{\lambda}_U$$

If $\hat{\lambda} \geq 1 + 3\delta$ we can find the set $L_2 = \{U : \hat{\lambda}_U \geq \hat{\lambda} - \delta^3/10; U \in \mathcal{O}_\delta\}$ in $O(m' + n\,\mathrm{poly}\{\delta^{-1}, \log n\})$ time using $O(n\delta^{-5})$ space where $m' = |\{(i,j)|\hat{x}_{ij} > 0\}|$.

Proof. We first observe that L_2 is a laminar family using Theorem 7.2.1 and $L_2 \subseteq L_1$. Second, observe that for any U we have $\sum_{(i,j):i,j\in U} \hat{x}_{ij} \leq \frac{1}{2}\sum_{i\in U}\sum_j \hat{x}_{ij} \leq \frac{1}{2}\sum_{i\in U} b_i = ||U||_b/2$. Therefore $\hat{\lambda} \leq \frac{3}{2}/(1 - \frac{\delta^2}{4}) < \frac{3}{2} + \delta^2$; the worst case gap between the vertex constraints and odd-set constraints of size up to $1/\delta$ still happen at size 3 for the said range of δ.

We maintain an estimate $\tilde{\lambda}$ of such that $\tilde{\lambda} - \frac{\delta^3}{100} < \hat{\lambda}_U \leq \tilde{\lambda} \leq \frac{3}{2} + \delta^2$. This estimate can be found using binary search (as described below).

Create a graph G_φ with $p_{ij} = \lfloor \varphi \hat{x}_{ij} \rfloor$ parallel edges between i and j where $\varphi = 50/\delta^4$ (this parameter can be optimized but we omit that in the interest of simplicity). This is an unweighted graph. This graph can be constructed in a single pass over $\{(i,j)\}$. We also "merge" all pairs of vertices i and j if p_{ij} exceeds 2φ. Moreover delete vertices i with $2\varphi/\delta$ edges – note that these vertices must have $b_i \geq \sum_j \hat{x}_{ij} > 1/\delta$ and cannot participate in any odd set in $\mathcal{O}_\delta$. This gives us a graph G_φ with at most $O(n\delta^{-5})$ edges.

Now for an odd $\ell \in [3, 1/\delta]$ and $\tilde{\lambda}$, create $G_\varphi(\ell, \tilde{\lambda})$ as follows: Let $q_i = \lfloor \varphi\tilde{\lambda}(1 - \delta^2\ell)b_i \rfloor$ for all i. Since $q_i > (1+\delta)\varphi b_i > \sum_j p_{ij}$ (because $\tilde{\lambda}$ is large) we can add a new

node s and add $q_i - \sum_j p_{ij}$ edges between s and i (for all i). This gives us a graph $G_\varphi(\ell, \tilde{\lambda})$ of size $O(n\delta^{-5})$ edges for all ℓ. Let $\kappa(\ell) = \lfloor \varphi\tilde{\lambda}(1-\delta^2\ell^2/2)\rfloor + \frac{12\ell}{\delta} + 1 < 2\varphi$. Now:

$$q_i - \kappa(\ell) \geq \varphi\tilde{\lambda}(1-\delta^2\ell) - 1 - \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) - \frac{12\ell}{\delta} - 1 = \frac{\varphi\tilde{\lambda}\delta^2\ell(\ell-2)}{2} - \frac{12\ell}{\delta} - 2$$

which is positive for $\varphi = 50/\delta^4$ and $\ell \geq 3$. Therefore $q_i > \kappa(\ell)$. We now show that for $\mathcal{C}ut(U)$ to be small for any odd set U we must have $||U||_b \leq 1/\delta$. Now for any odd set $U \in \mathcal{O}$ with $||U||_b > 1/\delta$:

$$\begin{aligned}
\mathcal{C}ut(U) \quad - \quad & \kappa(\ell) = \sum_{i\in U} q_i - 2\sum_{(i,j):i,j\in U} p_{ij} - \kappa(\ell) \\
\geq \quad & \sum_{i\in U}(\varphi\tilde{\lambda}(1-\delta^2\ell)b_i - 1) - 2\varphi\sum_{(i,j):i,j\in U}\hat{x}_{ij} - \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) - \frac{12\ell}{\delta} - 1 \\
\geq \quad & \varphi\tilde{\lambda}(1-\delta^2\ell)||U||_b - |U| - \varphi||U||_b - \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) - \frac{12\ell}{\delta} - 1 \\
& \qquad \text{(Since for any } U, \textstyle\sum_{(i,j):i,j\in U}\hat{x}_{ij} \leq ||U||_b/2) \\
\geq \quad & \varphi\tilde{\lambda}(1-\delta^2\ell)||U||_b - |U| - \varphi||U||_b - \varphi\tilde{\lambda} - \frac{12\ell}{\delta} - 1 \\
\geq \quad & \varphi\tilde{\lambda}(1-\delta)||U||_b - \varphi||U||_b - \varphi\tilde{\lambda} - \delta^2\varphi||U||_b \\
& \qquad \text{(Since } \ell \leq 1/\delta \text{ and } \delta^2\varphi||U||_b > |U| + \tfrac{12\ell}{\delta} + 1) \\
= \quad & \varphi\left(\tilde{\lambda}(1-\delta)||U||_b - \tilde{\lambda} - (1+\delta^2)||U||_b\right) \\
\geq \quad & \varphi\left((1+3\delta)(1-\delta)||U||_b - \frac{3}{2} - \delta^2 - (1+\delta^2)||U||_b\right) \\
& \qquad \text{(Since } 1+3\delta \leq \hat{\lambda} \leq \tilde{\lambda} \leq \tfrac{3}{2} + \delta^2) \\
> \quad & \varphi\left(2\delta(1-2\delta)||U||_b - \frac{3}{2} - \delta^2\right) > \varphi\left(2(1-2\delta) - \frac{3}{2} - \delta^2\right) > 0
\end{aligned}$$

where the last inequality follows from $||U||_b > 1/\delta$ and $\delta \in (0, \frac{1}{16}]$. Therefore no

odd-set with $||U||_b > 1/\delta$ satisfies $\mathcal{C}ut(U) \leq \kappa(\ell)$.

We now show Property 1, namely: If $\tilde{\lambda} - \frac{\delta^3}{100} < \hat{\lambda} \leq \tilde{\lambda}$, then (i) all sets in $L_2(\ell)$ have a cut which is at most $\kappa(\ell)$ and (ii) all odd sets of $G_\varphi(\ell, \tilde{\lambda})$ which do not contain s and have cut at most $\kappa(\ell)$ belong to $L_1(\ell)$. For part (i) for a set $U \in L_2(\ell)$ with $||U||_b = \ell$, note $|U| \leq ||U||_b = \ell$ and:

$$\begin{aligned}
\mathcal{C}ut(U) &= \sum_{i\in U} q_i - 2\sum_{(i,j):i,j\in U} p_{ij} \leq \sum_{i\in U} \varphi\tilde{\lambda}(1-\delta^2\ell)b_i - 2\varphi \sum_{(i,j):i,j\in U} \hat{x}_{ij} + |U|^2 \\
&\leq \varphi\tilde{\lambda}(1-\delta^2\ell)||U||_b - 2\varphi\hat{\lambda}_U\tilde{b}_U + \ell^2 \\
&\leq \varphi\tilde{\lambda}(1-\delta^2\ell)||U||_b - 2\varphi\left(\tilde{\lambda} - \frac{\delta^3}{100} - \frac{\delta^3}{10}\right)\tilde{b}_U + \ell^2 \\
&= \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) + \frac{11\delta^3\varphi\tilde{b}_U}{50} + \ell^2 = \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) + \frac{11\tilde{b}_U}{\delta} + \ell^2 \\
&\leq \varphi\tilde{\lambda}(1-\delta^2\ell^2/2) + \frac{12}{\delta} \leq \kappa(\ell) \quad (\text{since } \tilde{b}_U < ||U||_b = \ell \leq 1/\delta)
\end{aligned}$$

To prove part (ii) if $\mathcal{C}ut(U) \leq \kappa(\ell)$ then:

$$\begin{aligned}
\sum_{(i,j):i,j\in U} p_{ij} &= \frac{1}{2}\left(\sum_{i\in U} q_i - \mathcal{C}ut(U')\right) \geq \frac{1}{2}\left(\sum_{i\in U}\left(\varphi\tilde{\lambda}(1-\delta^2\ell)b_i - 1\right) - \kappa(\ell)\right) \\
&\geq \frac{1}{2}\left(\sum_{i\in U}\left(\varphi\tilde{\lambda}(1-\delta^2\ell)b_i - 1\right) - \varphi\tilde{\lambda}(1-\delta^2\ell^2/2)\right) - \frac{12\ell}{\delta} - 1 \\
&\geq \varphi\tilde{\lambda}\left(\left\lfloor\frac{||U||_b}{2}\right\rfloor - \frac{\delta^2||U||_b^2}{4}\right) + \frac{\varphi\tilde{\lambda}\delta^2}{4}\left(||U||_b - \ell\right)^2 - \frac{|U|}{2} - \frac{12\ell}{\delta} - 1 \\
&= \varphi\tilde{\lambda}\tilde{b}_U + \frac{\varphi\tilde{\lambda}\delta^2}{4}\left(||U||_b - \ell\right)^2 - \frac{|U|}{2} - \frac{12\ell}{\delta} - 1
\end{aligned}$$

But since $\tilde{\lambda} \geq \hat{\lambda} \geq \hat{\lambda}_U$ and $\varphi\tilde{b}_U\hat{\lambda}_U = \varphi\sum_{(i,j):i,j\in U}\hat{x}_{ij} \geq \sum_{(i,j):i,j\in U} p_{ij}$ we have

$$\varphi\tilde{\lambda}\tilde{b}_U \geq \varphi\hat{\lambda}_U\tilde{b}_U \geq \varphi\tilde{\lambda}\tilde{b}_U + \frac{\varphi\tilde{\lambda}\delta^2}{4}\left(||U||_b - \ell\right)^2 - \frac{|U|}{2} - \frac{12\ell}{\delta} - 1 \tag{7.5.1}$$

But that is a contradiction unless $||U||_b = \ell$, otherwise the quadratic term, $\frac{\varphi\tilde{\lambda}\delta^2}{4}\left(||U||_b - \ell\right)^2 \geq 12.5\delta^{-2}$ is larger than the negative terms which are at most $\frac{1}{2\delta} +$

$\frac{12}{\delta^2}+1$ in the RHS of Equation 7.5.1. Therefore $\mathcal{C}ut(U) \leq \kappa(\ell)$ for an odd-set implies $||U||_b = \ell$. But then Equation 7.5.1 implies (again using $|U| \leq ||U||_b = \ell$):

$$\varphi\hat{\lambda}_U\tilde{b}_U \geq \varphi\tilde{\lambda}\tilde{b}_U - \frac{\ell}{2} - \frac{12\ell}{\delta} - 1$$

Now $\tilde{b}_U \geq \frac{\ell}{3}(1-\frac{3\delta}{4})$ when $||U||_b = \ell \geq 3$; thus we have:

$$\begin{aligned}\hat{\lambda}_U &\geq \tilde{\lambda} - \frac{\ell}{2\varphi\tilde{b}_U} - \frac{12\ell}{\delta\varphi\tilde{b}_U} - \frac{1}{\varphi\tilde{b}_U} \\ &\geq \tilde{\lambda} - \frac{3\delta^4}{100(1-\frac{3\delta}{4})} - \frac{36\delta^3}{50(1-\frac{3\delta}{4})} - \frac{\delta^4}{50} > \tilde{\lambda} - \delta^3 \geq \hat{\lambda} - \delta^3\end{aligned}$$

in other words, $\mathcal{C}ut(U) \leq \kappa(\ell)$ for an odd-set implies $U \in L_1(\ell)$, as claimed in part(ii).

We now apply Lemma 7.5.1 to extract a collection $\mathcal{L}$ of odd-sets in $G_\varphi(\ell, \tilde{\lambda})$, not containing s and cut at most $\kappa(\ell)$ – such that any such set which is not chosen must intersect with some set in the collection $\mathcal{L}$.

If we have a maximal collection $\mathcal{L}$ then $\mathcal{L} \subseteq L_1(\ell)$ by part (ii) of Property 1. Due to Theorem 7.2.1, the intersection of two such sets $U_1, U_2 \in L_1(\ell)$ will be either empty or of size ℓ by laminarity – the latter implies $U_1 = U_2$. Therefore the sets in $L_1(\ell)$ are disjoint. Any $U \in L_2(\ell) - \mathcal{L}$ has a cut of at most $\kappa(\ell)$ using part (i) of Property 1 and therefore must intersect with some set in $\mathcal{L}$. This is impossible because $U \in L_2(\ell)$ implies $U \in L_1(\ell)$ and $\mathcal{L} \subseteq L_1(\ell)$ and we just argued that the sets in $L_1(\ell)$ are disjoint! Therefore no such U exists and $L_2(\ell) \subseteq \mathcal{L}$.

We now have a complete algorithm: we perform a binary search over the estimate $\tilde{\lambda} \in [1+3\delta, \frac{3}{2}+\delta^2]$, and we can decide if there exists a set $U \in L_2(\ell)$ in time $O(n\,\mathrm{poly}(\delta^{-1}, \log n))$ as we vary $\ell, \tilde{\lambda}$. This gives us $\tilde{\lambda}$. We now find the collections $\mathcal{L}$

for each ℓ and compute all $\hat{\lambda}_U$ exactly (either remembering the $\hat{x}_{ij}$ of the the edges stored in G_φ or by another pass over G). We can now return $\cup_\ell L_2(\ell)$. Observe that G_φ does not need to be constructed more than once; it can be stored and reused. The running time follows from simple accounting. □

In the remainder of the section we prove Lemma 7.5.1.

7.5.1 Proof of Lemma 7.5.1

Lemma 7.5.1. *Given an unweighted graph G with parameters κ, ϕ and a special node s, in time $O(n \operatorname{poly}(\delta^{-1}, \log n))$ we can identify a collection $\mathcal{L}$ of odd-sets which (i) do not contain s (ii) define cut of at most κ in G and (iii) every other odd set not containing s and with a cut less than κ intersects with a set in $\mathcal{L}$.*

Proof. The algorithm is given in Algorithm 28. First, consider the following:

Theorem 7.5.2 ([20, 57])**.** *Given a graph with n nodes and m edges (possibly with parallel edges), in time $O(m) + \tilde{O}(n\kappa^2)$ we can construct a weighted tree T that represents all min s–t cuts in G' of value at most κ. The nodes of this tree are subsets of vertices. The mincut of any pair of vertices that belong to the same subset (the same node in the tree T) is larger than κ and for any pair of vertices i, j belonging to different subsets (nodes in the tree T) the mincut is specified by the partition corresponding to the least weighted edge in the tree T between the two nodes that contain i and j respectively.*

Algorithm 28 Finding a maximal collection of odd-sets

1: $\mathcal{K} \leftarrow \emptyset$. Initially $G' = G$. The node $s \in V(G)$.

2: **repeat**

3: Assign the s duplicity $b_s = 1$ if $\sum_{i \in V(G')} b_i$ is odd. Otherwise let $b_s = 2$.

4: Construct a tree $\mathcal{T}$ that represents **all low** s–t **cuts** in G' using Theorem 7.5.2. The nodes of this tree $\mathcal{T}$ correspond to subsets of vertices of $V(G')$.

5: Make the vertex set containing s the root of $\mathcal{T}$ and **orient all edges towards the root**. The oriented edges represent an edge from a child to a parent. Let $D(e)$ indicate the set of descendant subsets of an edge e (including the child subset which is the tail of the edge, but not including the parent subset which is the head of the edge).

6: Using dynamic programming starting at the leaf, mark every edge as admissible/inadmissible based on the $\sum_{S \in D(e)} \sum_{i \in S} b_i$ over the descendant subsets of that edge being odd/even respectively.

7: Starting from the root s downwards, pick the edges e in parallel such that (c1) the weight of e (corresponding to a cut) is at most κ, (c2) $\sum_{S \in D(e)} \sum_{i \in S} b_i$ is odd and (c3) no edge e' on the path from e to s satisfies (i) and (ii). Let the odd-set U_e corresponding to this edge $e \in \mathcal{T}$ be $U_e = \cup_{S \in D(e)} S$.

8: If the odd-sets found are $U_{e_1}, \ldots, U_{e_g}$ then $\mathcal{K} \leftarrow \mathcal{K} \cup \{U_{e_1}, \ldots, U_{e_t}\}$. Observe that the sets U_{e_r} are disjoint and do not contain s.

9: Merge all vertices in $\bigcup_{r=1}^{g} U_{e_r}$ with s. Observe that for any set U that does not contain s and does not intersect any U_{e_r}, the cut $\mathcal{C}ut(U)$ is unchanged.

10: **until** no new odd set has been found in G'

11: **return** $\mathcal{L}$.

Lemma 7.5.3 (Implicit in [82]). *For any odd-set U in G with cut κ, there exists an edge (u, v) in the low min s-t cut tree $\mathcal{T}$ such that $u \in U, v \notin U$ and removing (u, v) from the tree results in two connected components of odd sizes. In addition, one of the components define an odd set with cut κ in the original graph G.*

Proof. (Of Lemma 7.5.3) Consider all the edges in the tree $\mathcal{T}$ that crosses the boundary of U. Removing each such edge from the tree results in two connected components where sizes of two components are both even or both odd. U can be represented as inclusion and exclusion of these sets. Therefore, if the size of these sets are all even, then $||U||_b$ has to be even. So there exists at least one edge (u, v) in the tree $\mathcal{T}$ such that its removal results in two connected components of odd sizes. U is an $u - v$ cut and therefore, the weight of (u, v) in the tree T is at most κ. Choosing the side that does not contain s, we obtain an odd set in the original graph. In addition, the corresponding cut size is less than κ. □

(Continuing with Proof of Lemma 7.5.1.) All that remains to be proven is that the loop in Algorithm 28 needs to be run only a few times. Suppose after t' repetitions $Q_{t'}$ is the maximum collection of disjoint odd-sets which are κ-attached and we choose $U_{e_1}, \ldots, U_{e_g}$ to be added to $\mathcal{L}$ in the $t' + 1$st iteration. We first claim that $|Q_{t'+1}| \leq g$. To see this we first map every odd-set in $Q_{t'+1}$ to an edge in the tree as specified by the existence proof in Lemma 7.5.3. This map need not be constructive – the map is only used for this proof. Observe that this can be a many to one map; i.e., several sets mapping to the same edge.

Now every edges $e_1, \ldots, e_g$ chosen in Algorithm 28 satisfy the property for all j: no edge e' on the path from the head of e_j (recall that the edges are oriented towards the root s) to s is one of the edges in our map. Because in that case we would have chosen that edge e' instead of e_j.

Therefore the sets in $Q_{t'+1}$ could not have mapped to any edges in the path towards s. Now, if a set in $Q_{t'+1}$ mapped to an edge e' which is a descendant of the tail of some e_j (again, the edges are oriented towards s) then this set intersects with our chosen U_{e_j} which is not possible.

Therefore any set in $Q_{t'+1}$ must have mapped to the same edges in the tree; i.e., $e_1, \ldots, e_g$. But then the vertex at the head of the edge belongs to the set in $Q_{t'+1}$. Therefore there can be at most g such sets. This proves $|Q_{t'+1}| \leq g$.

We next claim that $|Q_{t'+1}| \leq |Q_{t'}| - g$. Consider $Q' = Q_{t'+1} \cup \{U_{e_1}, \ldots, U_{e_g}\}$. Q' is a collection of disjoint odd-sets which define a cut of size κ in G after t' repetitions. Obviously $|Q'| = |Q_{t'+1}| + g$ and by the definition of $Q_{t'}$, $|Q'| \leq |Q_{t'}|$. Therefore, $|Q_{t'+1}| \leq |Q_{t'}| - g$.

Therefore, in the worst case, $|Q_{t'}|$ decreases by a factor $1/2$ and therefore in $O(\log n)$ iterations over this loop we would eliminate all odd-sets that define a cut of size κ in G'. $\square$

7.6 Initial Solutions

In this section we provide primal-dual approximation algorithms for both uncapacitated and capacitated b–Matching. The capacities b'_i, c'_{ij}, for vertices and edges respectively, need not be integral for this section. Each edge (i,j) has weight w'_{ij}. In the uncapacitated case $c'_{ij} = \infty$. The formulation LP27 captures the basic constraints which are sufficient for the purposes of this section – we are explicitly writing down a relaxation which omits non-bipartite constraints. The system LP28 is the dual of LP27. These formulations are undirected and we have only one variable for both $(i,j) = (j,i)$.

$$\begin{aligned}
&\hat{\beta} = \min \textstyle\sum_{(i,j)} w'_{ij} x_{ij} \\
&\tfrac{1}{b'_i} \textstyle\sum_j x_{ij} \leq 1 \qquad \forall i \\
&\tfrac{1}{c'_{ij}} x_{ij} \leq 1 \qquad \forall (i,j) \in E \\
&x_{ij} \geq 0
\end{aligned} \tag{LP27}$$

$$\begin{aligned}
&\hat{\beta} = \min \textstyle\sum_i p_i + \sum_{(i,j)} q_{ij} \\
&\tfrac{p_i}{b'_i} + \tfrac{p_j}{b'_j} + \tfrac{q_{ij}}{c'_{ij}} \leq w'_{ij} \quad \forall (i,j) \in E \\
&p_i, q_{ij} \geq 0
\end{aligned} \tag{LP28}$$

A simple primal-dual algorithm is provided in Algorithm 29. Observe that we maintain a feasible primal and a feasible dual solution. Observe that after the update, for any deleted edge we have $\frac{p_i}{b'_i} + \frac{p_j}{b'_j} \geq w'_{ij}$.

Definition 7.6.1. Let $\Upsilon = \sum_i p_i$. Define the increase in p_i, p_j due to the edge (i,j) to be the **direct** contribution of edge (i,j). If the edge (i,j) replaces $e_1, \ldots, e_\ell$ (possibly the last edge is replaced fractionally) then make two copies of the edge e_ℓ, one copy got deleted and the other copy stayed in the solution. Therefore without loss of generality define the **indirect** contribution of the edge (i,j) to be the **sum** of

Algorithm 29 Linear time single pass algorithm for capacitated b–Matching

1: We start with all $p_i = 0$. Throughout the algorithm we will maintain the invariant $p_i \geq 2\sum_j w'_{ij}x_{ij}$. Assume that we have some hypothetical vertex v which has 0 weight edges to every other node with $b_v = \infty$ and $x_{vj} = b_j$ for all j.

2: **for** each new edge $e = (i, j)$ **do**

(a) If $\frac{p_i}{b'_i} + \frac{p_j}{b'_j} \geq w'_{ij}$ then do nothing, otherwise:

(b) Let $x = \min\{c'_{ij}, b'_i, b'_j\}$.

(c) Delete the cheapest x (fractionally) edges incident to i (and same for j). In more detail: Order the edges $\{(i', j)|x_{i'j} \geq 0\}$ in increasing order of $w'_{i'j}$. Find $i(j)$ such that $\sum_{i'<i(j)} x_{i(j)j} < x$ and $\sum_{i'\leq i(j)} x_{i'j} \geq x$. Set $y_{i(j)j} \leftarrow \sum_{i'\leq i(j)} x_{i'j} - x$. For $i' > i(j)$ keep $x_{i'j}$ unchanged. For $i' < i(j)$ set $x_{i'j} = 0$.

(d) Set $x_{ij} = x$. Increase p_i, p_j to be at least $2\sum_j w'_{ij}x_{ij}, 2\sum_i w'_{ij}x_{ij}$ respectively.

the direct and indirect contributions of the edges $e_1, \ldots, e_\ell$.

The direct contribution of any edge (i, j) is at most $4w'_{ij}x_{ij}$ (two vertices, each of whose p_i value increases by at most $2w'_{ij}x_{ij}$). We increased p_i, p_j only when $\frac{p_i}{b'_i} + \frac{p_j}{b'_j} < w'_{ij}$. Since $p_i \geq 2\sum_j w'_{ij}x_{ij}$ (and likewise for p_j) before the edges incident on i (and some of the edges incident to j) were deleted; the direct contribution of the edges deleted when (i, j) was inserted is at most $\frac{1}{2}\left(2w'_{ij}x_{ij}\right)$. To see this, divide (i, j) and the deleted edges infinitesimally; for each infinitesimal copy of (i, j) with $x_{ij} = \Delta$. If an infinitesimal copy of (i, j) causes the deletion of e_1, e_2 (incident at i, j respectively, each with the same infinitesimal Δ_{ij} amount as (i, j)) then $w'_{ij} \geq 2(w'(e_1) + w'(e_2))$ because we deleted the cheapest edges. The direct contribution of these edges is $\Delta(2w'(e_1) + 2w'(e_2))$. Therefore the direct contribution of the edges deleted by the insertion of (i, j) is at most $\frac{1}{2}\left(2w'_{ij}x_{ij}\right) = w'_{ij}x_{ij}$. *Inductively*, the indirect contribution of edge (i, j) is also at most $2w'_{ij}x_{ij}$ using the facts that the weights of the (sets of) edges in a chain of deletions decrease geometrically by factor 2. Therefore $\sum_i p_i = \Upsilon \leq 6\sum_{(i,j)} w'_{ij}x_{ij}$. The above accounting of the charge is the "trail of the dead" analysis in [43], and can also be found in the analysis of call-admission algorithms [1]. Therefore if at the end we are left with a set S of edges; for the capacitated problem we set $q_{ij} = w'_{ij}y_{ij}$ for $(i, j) \in S$ and 0 otherwise. This is a feasible dual solution and observe that $\sum_i p_i + \sum_{i,j} q_{ij} \leq 7\sum_{(i,j)\in S} w'_{ij}x_{ij}$. But this is a feasible dual solution and therefore $\hat{\beta} \leq 7\sum_{(i,j)\in S} w'_{ij}x_{ij}$. Furthermore, observe that the solution either saturates and edge or a vertex at each step. Thus:

Theorem 7.6.2. *We can solve the capacitated b–Matching problem to an approximation factor 7 within the optimum fractional solution. Moreover the number of edges in the solution is* $\min\{m, B\}$.

For the uncapacitated case, the variables q_{ij} do not exist. Therefore $\sum_i p_i \leq 6\sum_{(i,j)\in S} w'_{ij}x_{ij}$. This gives a 6 approximation and we have at most $O(n)$ edges.

Theorem 7.6.3. *We can solve the uncapacitated b–Matching problem to an approximation factor 6 of the optimum fractional solution. Moreover the number of edges in the solution is* $O(n)$. *Furthermore setting* $y^{\dagger}_{ij} = 6x_{ij}$ *will give us a solution where* $\sum_{(i,j)} x^{\dagger}_{ij}w'_{ij} \geq \beta^*$ *and* $\sum_j y^{\dagger}_{ij} \leq 6b'_i$ *which gives us the initial solution for Algorithm 27 as well as the solution sought after in Lemma 7.3.2.*

7.7 Lagrangians and The Oracle

In this section we prove Lemma 7.3.2. Recall $\gamma = \sum_i y_i\tilde{b}_i + \sum_{U\in L} z_U\tilde{b}_U$ where $\tilde{b}_i = (1-4\delta)$ and $\tilde{b}_U = \left\lfloor \frac{||U||_b}{2} \right\rfloor - \delta^2||U||^2_b/4$.

Lemma 7.3.2. *If* $(1-4\delta)\beta^* \geq \beta$ *then (i) we always solve LP25 (and do not decrease* β*) and (ii) we can solve LP24 using* $O(1/\delta)$ *invocations of LP25 (for different* $\varrho \geq 0$*) and using the convex combination of two solutions. The solution for LP25 uses* $O(m)$ *time,* $O(n/\delta)$ *space, a single pass over the edges and outputs a solution with* $O(n)$

non-zero edges. β^* is defined in LP22 (repeated for convenience).

$$\mathcal{L}(\mathbf{y}', \varrho) = \sum_{(i,j)} w_{ij} x'_{ij} - \varrho \left(\sum_{(i,j)} x'_{ij} (y_i + y_j + \sum_{U \in L; i,j \in U} z_U) \right) \geq \beta - \varrho\gamma \tag{LP25}$$

$$\mathcal{P} : \begin{cases} \sum_j x'_{ij} \leq 6b_i & \forall i \in V \\ x'_{ij} \geq 0 \end{cases}$$

$$\sum_{(i,j)} \tilde{x}_{ij} (y_i + y_j + \sum_{U \in L; i,j \in U} z_U) \leq \gamma$$

$$\mathcal{P}[\beta] : \begin{cases} \sum_{(i,j)} w_{ij} x_{ij} \geq (1-\delta)\beta \\ \mathbf{x} \in \mathcal{P} \end{cases}$$

(LP24)

$$\beta^* = \max \sum_{(i,j)} w_{ij} x_{ij}$$

$$\begin{aligned} &\textstyle\sum_j x_{ij} \leq b_i && \forall i \in V \\ &\textstyle\sum_{(i,j): i,j \in U} x_{ij} \leq \lfloor ||U||_b / 2 \rfloor && \forall U \in \mathcal{O} \\ &x_{ij} \geq 0 \end{aligned}$$

(LP22)

Proof. Consider the optimum solution of LP22 given by $\{x^*_{ij}\}$ and define $x'_{ij} = (1-4\delta)x^*_{ij}$. This solution satisfies $\sum_{(i,j)} w_{ij} x'_{ij} = (1-4\delta)\beta^*$ and $\sum_j x'_{ij} \leq (1-4\delta)b_i = \tilde{b}_i$ and $\sum_{(i,j):i,j \in U} x'_{ij} \leq (1-4\delta) \left\lfloor \frac{||U||_b}{2} \right\rfloor$ for any U. In particular we satisfy $\sum_{(i,j):i,j \in U} x'_{ij} \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor - \frac{\delta^2 ||U||_b^2}{4} = \tilde{b}_U$.

Since y_i, z_U are non-negative, $\{x'_{ij}\}$ satisfies the linear combination that

$$\sum_i y_i \sum_j x'_{ij} + \sum_{U \in L} z_U \sum_{(i,j); i,j \in U} x'_{ij} \leq \sum_i y_i \tilde{b}_i + \sum_{U \in L} z_u \tilde{b}_U \leq \sum_i y_i \tilde{b}_i + \sum_{U \in \mathcal{O}_\delta} z_u \tilde{b}_U = \gamma$$

The left hand side rearranges to: $\sum_{(i,j)} x'_{ij} (y_i + y_j + \sum_{U \in L; i,j \in U} z_U)) \leq \gamma$ which implies that LP24 is feasible as long as $\beta \geq (1-4\delta)\beta^*$. And since $\varrho \geq 0$ we have: $\mathcal{L}(\mathbf{y}', \varrho) \geq \beta^* - \varrho\gamma$ which implies that LP25 is feasible with the stronger constraint that $\sum_j y'_{ij} \leq b_i$.

Given an L we can precompute and store the quantities $(y_i + y_j + \sum_{U \in L; i,j \in U} z_U)$ for all (i, j) irrespective of in E or otherwise. This can be done in n/δ time and space since each set of L is at most of size $O(1/\delta)$ and therefore we affect at most $O(n/\delta)$ edges. The problem LP25 reduces to finding a b–Matching (ignoring the odd set constraints) using "effective weights" instead of w_{ij}. Suppose we had a single pass $O(m)$ time 6 approximation algorithm using $O(n)$ edges, then we can find that solution $\{x^{\dagger}_{ij}\}$ and simply set $x'_{ij} = 6x^{\dagger}_{ij}$. The approximation factor guarantees that the contribution of x'_{ij} is more than $(1 - 4\delta)\beta^* - \varrho\gamma$. Note that the solution $\{x'_{ij}\}$ only needs to satisfy $\sum_j x_{ij} \leq 6b_i$. We show in Theorem 7.6.3, in Section 7.6, how to find such a 6 approximation (along with how to compute an initial solution), but any $O(m)$ time c-approximation will work (making $\lambda_0 = 2c$ in Algorithm 27). Lemma 7.3.2 follows from applying Lemma 6.2.3 with $q = 1$ and $r = \delta$. □

7.8 Rounding Fractional Uncapacitated b-Matchings

In this section, we discuss a space and time efficient rounding algorithm.

Theorem 7.1.4. *Given a fractional b-matching* $\mathbf{x}$ *which satisfies* LP22($\mathbf{b}$) *(parameterized over* $\mathbf{b}$*) where* $|\{(i,j)|x_{ij} > 0\}| = m'$*, we find an integral b–Matching of weight at least* $(1 - 2\delta)\sum_{(i,j)} w_{ij}x_{ij}$ *in* $O(m'\delta^{-3}\log(1/\delta))$ *time and* $O(m'/\delta^2)$ *space.*

The algorithm is given in Algorithm 30. We prove the following lemma:

Algorithm 30 Rounding a fractional b–Matching (Part I)

1: First Phase: **Removing edges with large multiplicities** $t = \lceil 2/\delta \rceil$.

(a) If $x_{ij} \geq t$ add $\hat{x}_{ij}^{(0)} = \lfloor x_{ij} \rfloor - 1$ copies of (i, j) to $\mathcal{M}^{(0)}$.

(b) Set $x_{ij}^{(1)} = 0$ if $x_{ij} \geq t$ and $x_{ij}^{(1)} = x_{ij}$ otherwise.

(c) Let $b_i^{(1)} = \min \left\{ b_i - \sum_j \hat{x}_{ij}^{(0)}, \lceil \sum_j x_{ij}^{(1)} \rceil + 1 \right\}$.

2: Second Phase: **Subdividing large capacity vertices.**

(a) **While** there exists a vertex i s.t. $\sum_j x_{ij}^{(1)} \geq 3t$ **do**

(i) Observe that given a set of numbers $q_1, \ldots, q_k$ such that each $q_j \leq 1$ and $\sum_j q_j = Y \geq 3$; we can easily partition the set of numbers such that each partition S satisfies $1 \leq \sum_{j \in S} q_j \leq 3$.

(ii) Order the vertices adjacent to i arbitrarily. Select the prefix S in that order such that the sum is between t and $2t$ (each edge is at most t from Step 1b). Create a new copy i' of i with this prefix and $x_{i'j}^{(1)} = x_{ij}^{(1)}$ for $j \in S$ and delete the edges from S incident to i.

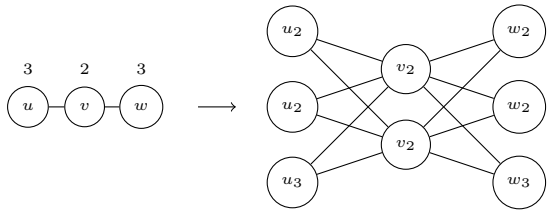
(b) If no copies of i were created then $b_i^{(2)} = b_i^{(1)}$. For every new i' (corresponding to i) created from the partition S (which may have now become S' with subsequent splits), assign $b_{i'}^{(2)} = \lfloor \sum_{j \in S'} x_{ij}^{(1)} \rfloor$. Note $b_i^{(2)} \leq 3t$ for all vertices. We now have a vertex set $V^{(2)}$. Set $x_{ij}^{(2)} = (1 - \delta) x_{ij}^{(1)}$ for $i, j \in V^{(2)}$.

Algorithm 30 Rounding a fractional b–Matching (Part II)

1: Third Phase: **Reduction to weighted matching.**

(a) For each $i \in V^{(2)}$ with $b_i^{(2)}$, create $i(1), i(2), \cdots, i(b_i^{(2)})$.

(b) For each edge (i, j), create a complete bipartite graph between $i(1), i(2), \cdots$ and $j(1), j(2), \cdots$ with every edge having weight w_{ij}. Let this new graph be $G^{(3)}$. As an example:



(c) Run any fast approximation for finding a $(1 - \epsilon)$-approximate maximum weighted matching in $G^{(3)}$ let this matching be $\mathcal{M}^{(3)}$.

2: Final Phase: **Output:** Observe that (i) Matching $\mathcal{M}^{(3)}$ implies a b–matching $\mathcal{M}^{(2)}$ in $G^{(2)}$ of same weight (merge edges). (ii) Matching $\mathcal{M}^{(2)}$ implies a b–matching $\mathcal{M}^{(1)}$ in $G^{(1)}$ of same weight (merge vertices). Output $\mathcal{M}^{(0)} \cup \mathcal{M}^{(1)}$.

Lemma 7.8.1. *(First Phase and the Output Phase) Suppose that all vertex constraints are satisfied and* $\sum_j x_{ij} \leq b_i - 1$*. Then, for any odd set* U *that contains* i*, the corresponding odd set constraint is satisfied. The fractional solution* $\{x_{ij}^{(1)}\}$ *obtained in the first phase of Algorithm 30 is feasible for* $\mathrm{LP22}(\mathbf{b^{(1)}})$ *— and a an integral* $\mathcal{M}^{(1)}$ *which is a* $(1-2\delta)$*-approximation of* $\mathrm{LP22}(\mathbf{b^{(1)}})$ *can be output along with* $\mathcal{M}^{(0)}$ *to satisfy Theorem 7.1.4.*

Proof. For any $U \in \mathcal{O}, i \in U$, $\sum_{i',j \in U} x_{i'j} \leq \frac{1}{2} \sum_{i' \in U} \sum_j x_{i'j} \leq \frac{1}{2}((\sum_{i' \in U} b_{i'}) - 1) = \frac{||U||_b - 1}{2} = \left\lfloor \frac{||U||_b}{2} \right\rfloor$. Thus it follows that any vertex which has an edge incident to it in $\mathcal{M}^{(0)}$ cannot be in any violated odd-set in $\mathrm{LP22}(\mathbf{b^{(1)}})$. Then any violated odd-set in $\mathrm{LP22}(\mathbf{b^{(1)}})$ with respect to $\{x_{ij}^{(1)}\}$ must also be a violated odd-set in $\mathrm{LP22}(\mathbf{b})$; contradicting the fact that we started with a $\{x_{ij}\}$ is feasible for $\mathrm{LP22}(\mathbf{b})$. Now $\mathcal{M}^{(0)} \cup \mathcal{M}^{(1)}$ is feasible since both are integral and we know that $b_i^{(1)} \leq b_i - \sum_j \hat{x}_{ij}^{(0)}$. Observe that $w(\mathcal{M}^{(0)}) \geq (1-\delta) \sum_{(i,j) \in E} w_{ij} \left(x_{ij} - x_{ij}^{(1)} \right)$ where $w(\mathcal{M}^{(0)}) = \sum_{(i,j) \in E} \hat{x}_{ij}^{(0)} w_{ij}$. Therefore if $w(\mathcal{M}^{(1)}) \geq (1-2\delta) \sum_{(i,j) \in E} w_{ij} x_{ij}^{(1)}$ then $w(\mathcal{M}^{(0)}) + w(\mathcal{M}^{(1)})$ is at least $(1-2\delta) \sum_{(i,j) \in E} w_{ij} x_{ij}$ as desired. □

Lemma 7.8.2. *(Second Phase) If* $\{x_{ij}^{(1)}\}$ *satisfies* $\mathrm{LP22}(\mathbf{b^{(1)}})$ *over* V*, then* $\{x_{ij}^{(2)}\}$ *satisfies* $\mathrm{LP22}(\mathbf{b^{(2)}})$ *over* $G^{(2)}$*. Moreover* $\sum_{i,j} w_{ij} x_{ij}^{(2)} = (1-\delta) \sum_{i,j} w_{ij} x_{ij}^{(1)}$

Proof. Observe that any vertex which participates in any split produces vertices which have (fractionally) at least t edges. After scaling we have $(1-\delta) \sum_j x_{ij}^{(1)} \leq \sum_j x_{ij}^{(1)} - \delta t \leq \sum_j x_{ij}^{(1)} - 2 \leq b_i^{(2)} - 1$ from the definition of $b^{(2)}$ in line (3b) of Algorithm 30. Therefore the new vertices cannot be in any violated vertex or set constraint; from the

first part of Lemma 7.8.1 (now applied to $\mathrm{LP22}(\mathbf{b^{(1)}})$ instead of $\mathrm{LP22}(\mathbf{b})$). Therefore the Lemma follows. $\square$

Finally It is easy to see that any **integral** b–Matching in $G^{(2)}$ has an integral matching in $G^{(3)}$ of the same weight and vice versa — moreover given a matching for $G^{(3)}$ the integral b–Matching for $G^{(2)}$ can be constructed trivially. Also, the number of edges in $G^{(3)}$ is at most $O(\delta^{-2}m')$ since each vertex in $G^{(2)}$ is split into $O(\delta^{-1})$ vertices in $G^{(3)}$. We are guaranteed a maximum b–Matching in $G^{(2)}$ of weight at least $\sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)}$ since $\{x_{ij}^{(2)}\}$ satisfies $\mathrm{LP22}(\mathbf{b^{(2)}})$ over $G^{(2)}$. Therefore we are guaranteed a matching of the same weight in $G^{(3)}$. Now, we use the approximation algorithm in [34, 35] which returns a $(1-\delta)$-approximate maximum weighted matching in $G^{(3)}$ in $O(m'\delta^{-3}\log(1/\delta))$ time and space. From the $(1-\delta)$-approximate maximum matching we can construct a b–Matching in $G^{(2)}$ of the same weight (and therefore a a b–Matching $\mathcal{M}^{(1)}$ in $G^{(2)}$ of the same weight). Theorem 7.1.4 follows.

Chapter 8

Application II: Capacitated b-Matching

Chapter Outline: In this chapter, we present algorithms for the capacitated b–Matching problems in general graphs. We reduce the capacitated b–Matching problem into the uncapacitated b–Matching problem and apply the algorithm in Chapter 7. However, the size of the graph increases as a result of the reduction and we apply nontrivial modifications to the algorithm. Again, for the simplicity of analysis, we use the insertion-only model in this chapter.

8.1 The Standard LP Formulation and Results

In this chapter we provide algorithms for finding fractional as well as integral solutions for the capacitated b-matching problem in general weighted graphs. The problem is

defined as follows:

Definition 8.1.1. [86, Chapters 32 & 33] In the **Capacitated b–Matching** problem we have an additional restriction for every edge $(i,j) \in E$ that $x_{ij} \leq c_{ij}$ where c_{ij} are also given in the input (also assumed to be an integer in $[0, \operatorname{poly} n]$). A problem with $c_{ij} = 1$ for all $(i,j) \in E$ is also referred to as an "unit capacity" or "simple" b–Matching problem in the literature.

The standard exact linear programming formulation for capacitated b–Matching is similar to the LP for uncapacitated b–Matching (LP22). LP29 is the standard LP. LP29 also has an integral optimum solution when b_i and c_{ij} are integers (See also [86, Chapter 32]).

Definition 8.1.2 (Volume of Sets & Odd-Sets for Capacitated b–Matching)**.** Given a graph $G = (V, E, c)$, with $|V| = n$ and $|E| = m$, and non-negative values b_i for each $i \in V$, define the **volume** of a set $U \subseteq V$ to be $||U||_{b,c} = \sum_{i \in U} b_i + \sum_{i \in U} \sum_{j \in V \setminus U} c_{ij}$. Define $\mathcal{O} = \{U \mid ||U||_{b,c} \text{ is odd } \}$ and $\mathcal{O}_\delta = \{U \mid U \in \mathcal{O}; ||U||_{b,c} \leq 1/\delta\}$.

$$
\begin{aligned}
\beta^* = \text{LP29}(\mathbf{b}, \mathbf{c}) = \quad & \max \textstyle\sum_{(i,j)} w_{ij} x_{ij} \\
& \textstyle\sum_j x_{ij} \leq b_i && \forall i \in V \\
& \textstyle\sum_{(i,j): i,j \in U} x_{ij} \leq \lfloor ||U||_{b,c}/2 \rfloor && \forall U \in \mathcal{O} \\
& 0 \leq x_{ij} \leq c_{ij}
\end{aligned}
\tag{LP29}
$$

Statement of Results and Roadmap: Theorems 8.1.3 and 8.1.4 summarize the results for capacitated b–Matching problem. Note that the restriction $\sum_{(i,j)\in\hat{E}} w_{ij} \leq T\beta^*$ of Theorem 8.1.3 is explicitly used in Theorem 8.1.4. Section 8.2 present an overview of the algorithm. Section 8.3 and Section 8.3.2 prove Theorem 8.1.3. Section 8.4 proves Theorem 8.1.4.

Theorem 8.1.3 (Fractional Capacitated b–Matching)**.** *For any $0 < \delta \leq 1/16$, we find a $(1-14\delta)$-approximate fractional weighted capacitated b-Matching using $O(\min\{m, B\,\mathrm{poly}\{\delta^{-1}, \ln n\}\})$ additional "work" space where $B = \sum_i b_i$ and making $R = O(\delta^{-4}(\ln^2(1/\delta))\ln n)$ passes over the list of edges. The algorithm runs in time $O(mT + \min\{B, m\}\,\mathrm{poly}\{\delta^{-1}, \ln n\})$ where $T = O(\delta^{-5}(\ln^2(1/\delta))\ln n)$. The algorithm returns a solution $\{\hat{x}_{ij}\}$ such that the subgraph $\hat{E} = \{(i,j)|(i,j) \in E, \hat{x}_{ij} > 0\}$ satisfies $\sum_{(i,j)\in\hat{E}} w_{ij} \leq R\beta^*$ where β^* is the weight of the optimum capacitated b–Matching.*

Theorem 8.1.4 (Integral Capacitated b–Matching)**.** *Given a feasible fractional capacitated b–Matching $\mathbf{x}$ such that the optimum solution is at most β^* and $\sum_{(i,j)\in\hat{E}} w_{ij} \leq R\beta^*$ where $\hat{E} = \{(i,j)|x_{ij} > 0\}$, we can find an integral b-matching of weight at least $(1-\delta)\sum_{(i,j)} w_{ij}x_{ij} - \delta\beta^*$ in $O(m'R\delta^{-3}\ln(R/\delta))$ time and $O(m'/\delta^2)$ space where $m' = |\hat{E}|$. If the fractional solution is $(1-14\delta)$-approximate then we have a $(1-16\delta)$-approximate integral solution.*

8.2 Algorithm Overview

Assume that the graph is presented as a read only list $\langle \ldots, (i, j, w_{ij}, c_{ij}), \ldots \rangle$ in arbitrary order. We are given a graph $G = (V, E)$ with $|V| = n$, $|E| = m$ and $B = \sum_i b_i$. We also assume that the working space is $\tilde{O}(B)$ rather than $\tilde{O}(n)$ because it is possible for an approximation b–Matching to have B edges. For example, suppose that G is a complete unweighted graph with $b_i = n$ for all i and $c_{ij} = 1$ for all (i, j). Then, the optimal solution or any constant-factor approximation solution requires $O(B) = O(n^2)$ edges. Our approach will be to use a reduction to the uncapacitated b–Matching problem while maintaining the following invariant:

Invariant 1. During the execution of the primal dual algorithm we will maintain that $x_{ij} \leq c_{ij}$ (for any current solution as well as any updates).

Assuming that the invariant holds, we can reduce the capacitated b–Matching problem into an equivalent b–Matching problem. This connection is known in the literature (see [86, Chapter 32]) and has been exploited before; however we will have differences with the existing approaches, which will be described in the footnotes. Intuitively, the graph obtained from this reduction can be viewed as a "long code" of the capacitated graph, and we will run Algorithm 27 on that encoding. We then indicate which substeps of Algorithm 27 are modified to ensure the invariant. Note that a primal candidate is infeasible and thus, it can violate the invariant even if the invariant is written as a constraint. We begin with the following:

Definition 8.2.1. Given a graph $G = (V, E)$ with vertex and edge capacities. Con-

sider subdividing each edge $e = (i, j)$ into $(i, p_{ij,i}), (p_{ij,i}, p_{ij,j}), (p_{ij,j}, j)$ where $p_{ij,i}, p_{ij,j}$ are new additional vertices with capacity $b^c_{p_{ij,i}} = b^c_{p_{ij,j}} = c_{ij}$. For $i \in V$ set $b^c_i = b_i$. We use the weights $\frac{1}{2}w_{ij}, 0, \frac{1}{2}w_{ij}$ for $(i, p_{ij,i}), (p_{ij,i}, p_{ij,j}), (p_{ij,j}, j)$ respectively. Denote this graph as $\textsc{Long}(G)$. Let the vertex set of $\textsc{Long}(G)$ be V_L. We use w^c_{ij} for the edge weights and $||U||_{b,c} = \sum_{i \in U} b^c_i$ for $U \subseteq V_L$ to indicate the volume of a set U in $\textsc{Long}(G)$. The odd-sets in $\textsc{Long}(G)$ are $\mathcal{O}_L$. Likewise define $\mathcal{O}_{\delta L} = \{U \in \mathcal{O}_L, ||U||_{b,c} \text{ is odd }\}$. The notation distinguishes G and $\textsc{Long}(G)$ since they will be used simultaneously.

Definition 8.2.2. Given an assignment $\mathbf{x} = \{x_{ij}\}$ over G such that $x_{ij} \leq c_{ij}$ for all $(i, j) \in E$ we define $\textsc{Long}(\mathbf{x}) = \{x^c_{i'j'}\}$ over $\textsc{Long}(G)$ as follows: we set $x^c_{i,p_{ij,i}} = x^c_{p_{ij,j},j} = x_{ij}$ and $x^c_{p_{ij,i},p_{ij,j}} = c_{ij} - x_{ij}$.

Likewise given an assignment $\mathbf{x}^c = \{x^c_{i'j'}\}$ over $\textsc{Long}(G)$ such that $x^c_{i,p_{ij,i}} = x^c_{p_{ij,j},j}$ and $x^c_{i,p_{ij,i}} + x^c_{p_{ij,i},p_{ij,j}} = c_{ij}$ we define $\textsc{Short}(\mathbf{x}^c)$ over G as follows: we set $x_{ij} = x^c_{i,p_{ij,i}}$. As long as the capacity constraints are met initially by $\mathbf{x}, \mathbf{x}^c$ note that $\textsc{Short}(\textsc{Long}(\mathbf{x})) = \mathbf{x}$ and $\textsc{Long}(\textsc{Short}(\mathbf{x}^c)) = \mathbf{x}^c$.

Note that Invariant 1 implies that $\mathbf{x}$ is non-negative vector if and only if $\textsc{Long}(\mathbf{x})$ is non-negative. It is easy to see that the capacitated b–Matching in G is equivalent to (uncapacitated) b–Matching in $\textsc{Long}(G)$.

Theorem 8.2.3. *[86, Implicit in proof of Theorem 32.2, Vol A, pages 564–565]*

$\textsc{Long}(\mathbf{x})$ *is feasible for LP31 on* $\textsc{Long}(G)$ *iff* $\mathbf{x}$ *is feasible for LP30 on* G.

$$\begin{aligned}
&\textstyle\sum_{i,j\in U} x_{ij} \le \left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor && \forall U \subset V \\
&\textstyle\sum_{j} x_{ij} \le b_i && \forall i \in V \\
&x_{ij} \le c_{ij} && \forall (i,j) \in E \\
&x_{ij} \ge 0 && \forall i,j \in V
\end{aligned} \tag{LP30}$$

$$\begin{aligned}
&\textstyle\sum_{i,j\in U} x^c_{ij} \le \left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor && \forall U \in \mathcal{O}_L \\
&\textstyle\sum_{j} x^c_{ij} \le b^c_i && \forall i \in V_L \\
&x^c_{ip_{ij,i}} + x^c_{p_{ij,i}p_{ij,j}} = c_{ij} && \forall (i,j) \in E \\
&x^c_{p_{ij,i}p_{ij,j}} + x^c_{p_{ij,j}j} = c_{ij} && \forall (i,j) \in E \\
&x^c_{ij} \ge 0 && \forall i,j \in V_L
\end{aligned} \tag{LP31}$$

Corollary 8.2.4. *If* $\beta^* = \max \sum_{i,j} x_{ij} w_{ij}$ *subject to (LP30) and* $\beta' = \max \sum_{i,j} x^c_{ij} w^c_{ij}$ *subject to (LP31) then* $\beta^* = \beta'$. *This is a consequence of the special weight sequence*[1] $\frac{1}{2}w_{ij}, 0, \frac{1}{2}w_{ij}$. *Thus we need to* $\max \sum_{i,j} x^c_{ij} w^c_{ij}$ *subject to (LP31).*

Now suppose we run Algorithm 27 on page 145 on $\textsc{Long}(G)$ staring from the formulation LP32 (instead of LP23, and using m instead of n in determining α_t; note

[1] This is different from the weights w_{ij}, w_{ij}, w_{ij} used in [86, Theorem 32.4, Vol A, page 567], which showed that capacitated b–Matching reduced to uncapacitated b–Matching with a "constant shift" in the objective function of $\sum_{(i,j)\in E} c_{ij} w_{ij}$. Introducing the shift at the top level of a primal-dual algorithm means that the substeps have to be more accurate.

$m \leq n^2$) and solving appropriate subproblems in lines 11-13.

$$\mathbf{A}_L^c : \begin{cases} \sum_j x_{ij}^c \leq \lambda \tilde{b^c}_i & \forall i \in V_L \qquad \text{where } \tilde{b^c}_i = (1-4\delta) b_i^c \\ \displaystyle\sum_{i,j \in U} x_{ij}^c \leq \lambda \tilde{b^c}_U & \forall U \in \mathcal{O}_{\delta L} \qquad \text{where } \tilde{b^c}_U = \left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor - \frac{\delta^2 ||U||_{b,c}^2}{4} \end{cases}$$

$$\mathcal{Q}^c[\beta] : \begin{cases} \displaystyle\sum_{i,j} w_{ij}^c x_{ij}^c \geq (1-\delta)\beta \\ \mathcal{Q}^c : \begin{cases} x_{ip_{ij,i}}^c + x_{p_{ij,i}p_{ij,j}}^c = b_{p_{ij,i}}^c & \forall (i,j) \in E \\ x_{p_{ij,i}p_{ij,j}}^c + x_{p_{ij,j}j}^c = b_{p_{ij,j}}^c & \forall (i,j) \in E \\ \sum_j x_{ij}^c \leq \lambda_0^c b_i^c / 2 & \forall i \in V \text{ (\textbf{not} } V_L\text{); note } b_i^c = b_i \text{ for } i \in V \\ x_{ij}^c \geq 0 \end{cases} \end{cases}$$

(LP32)

Differences between LP32 and LP23 raises two questions:

1. *How do we compute a dual candidate when we cannot even enumerate all the constraints?* Long(G) has $O(m)$ vertices rather than n vertices and therefore, we cannot store the vertices in the working space. We answer this question by proving that *most violated odd sets consist of vertices in G and edge-induced vertices $p_{ij,i}$ such that $x_{ip_{ij,i}}^c \neq 0$.*

2. *How do we maintain Invariant 1 or equivalently how do we maintain $\mathbf{x}^c \in \mathcal{Q}^c$?* We approximate capacitated b–Matching in G rather than b–Matching in Long(G). This guarantees that two additional constraints in $\mathcal{Q}^c$. However, this is not sufficient. $x_{p_{ij,i}p_{ij,j}}^c < 0$ if $x_{ij} > c_{ij}$, which is equivalent to $x_{ip_{ij,i}}^c > c_{ij}$. In order to avoid this problem, we replace a simple oracle of multiplying a constant-factor approximation with a multi-pass semi-streaming algorithm.

8.3 Finding a Fractional Solution in (near) Linear Time

Observe that the entire proof of Theorem 7.3.3 holds provided:

(a) β is not decreased below $(1 - O(\delta))\beta^*$ and we solve:

$$\sum_{i,j} \tilde{x}^c_{ij}(y^c_i + y^c_j + \sum_{U \in L^c; i,j \in U} z^c_U) \leq \gamma^c \qquad \text{subject to } \tilde{\mathbf{x}}^c \in \mathcal{Q}^c[\beta] \tag{LP33}$$

(instead of LP24) while preserving Invariant 1, where $\gamma^c = \sum_{i \in V_L} y^c_i \tilde{b^c}_i + \sum_{U \in L^c} z^c_U \tilde{b^c}_U$. We label the laminar family as $\mathcal{L}^c$ here.

(b) We have an initial solution where λ^c_0 and $\beta^c_0 \geq \beta^*$ are not too large (again preserving Invariant 1). We show that $\lambda^c_0 = O(\ln \frac{1}{\delta})$ and $\beta^c_0 \leq O(\beta^* \ln \frac{1}{\delta})$.

The number of invocations of LP33 will be $R' = O(\lambda^c_0 \delta^{-4} \ln n)$ – exactly along the lines of the proof of Theorem 7.1.3. We begin with the following:

Definition 8.3.1. Denote $\eta_{i'j'} = y^c_{i'} + y^c_{j'} + \sum_{U \in \mathcal{L}; i',j' \in U} z^c_U$ for an edge (i', j') in $\textsc{Long}(G)$. For an edge $(i, j) \in E$ (in G) define $a_{ij} = \eta_{i,p_{ij,i}} + \eta_{p_{ij},j} - \eta_{p_{ij,i}p_{ij,j}}$. Define $K = \sum_{(i,j) \in E} \eta_{p_{ij,i}p_{ij,j}} c_{ij}$.

Theorem 8.3.2. *If $\beta \leq (1 - 4\delta)\beta^*$ a solution to LP34 always exists*

$$\sum_{(i,j) \in E} a_{ij}\tilde{x}_{ij} \leq \gamma^c - K \qquad \textit{subject to } \tilde{\mathbf{x}} \in \begin{cases} \sum_{i,j} w_{ij}x_{ij} \geq \beta & \\ x_{ij} \leq c_{ij} & \forall (i,j) \in E \\ \sum_j x_{ij} \leq b_i & \forall i \in V \\ x_{ij} \geq 0 & \end{cases} \tag{LP34}$$

where $K = \sum_{(i,j)\in E} \eta_{p_{ij,i}p_{ij,j}} c_{ij}$ *as in Definition 8.3.1. Moreover, if* $a_{ij} = \eta_{i,p_{ij,i}} + \eta_{p_{ij},j} - \eta_{p_{ij,i}p_{ij,j}} \geq 0$ *for all* $(i,j) \in E$*, and LP34 is feasible then we can solve LP33*

$$\sum_{(i',j')} \tilde{x}^c_{i'j'} \eta_{i',j'} \leq \gamma^c \qquad \textit{subject to } \tilde{\mathbf{x}}^c \in \mathcal{Q}^c[\beta] \tag{LP33}$$

in $O(\frac{m}{\delta}\ln(\frac{1}{\delta}))$ *time using* $O(\min\{m,B\}\cdot\frac{1}{\delta}\ln(\frac{1}{\delta}))$ *space and* $\mathfrak{q} = O(\ln(\frac{1}{\delta}))$ *passes over the list of edges of* G*. Moreover if we define* $S = \{(i,j) \in E | \textsc{Short}(\tilde{\mathbf{x}}^c)_{ij} > 0\}$ *then* $\sum_{(i,j)\in S} w_{ij} \leq \mathfrak{q}\beta^*$ *and* $|S| \leq O(\min\{m,B\}\ln\frac{1}{\delta})$*. This also provides us with an initial solution with* $\lambda_0 = O(\ln\frac{1}{\delta})$ *and* $\beta_0 = O(\beta^*\ln\frac{1}{\delta})$*.*

Lemma 8.3.3. *If* $\lambda > 1 + 8\delta$ *(otherwise the algorithm has stopped) and* $\mathbf{x}^c \in \mathcal{Q}^c$ *(will be an invariant) some* $U \in \mathcal{O}_\delta(V')$ *contains any* $p_{ij,i}$ *such that either (i) both* $i,j \notin U$ *or (ii)* $x^c_{ip_{ij,i}} = x^c_{p_{ij,j}j} = 0$*, then* $\lambda_U < \lambda - \delta^2$*. As a consequence of (i),* $a_{ij} = \eta_{ip_{ij,i}} + \eta_{p_{ij,j}j} - \eta_{p_{ij,i}p_{ij,j}} \geq 0$*. As a consequence of (ii), we can compute the* z_U *in time* $O(m'\,\mathrm{poly}\{\delta^{-1},\log n\})$ *where* $m' = |\{(i,j)|\textsc{Short}(\mathbf{x}^c)_{ij} \neq 0\}|$*.*

Theorem 8.3.2 and Lemma 8.3.3 guarantee that $R' = O(\lambda_0^c\delta^{-4}\ln n)$ invocations of LP33 will provide us a $(1-14\delta)$ approximation. It takes $R = O(\delta^{-4}(\ln\frac{1}{\delta})^2\ln n)$ passes over the input. We will have $|\hat{E}| = m' = O(R\min\{m,B\})$ and therefore the overall running time will not exceed $O(mT + \min\{B,m\}\,\mathrm{poly}\{\delta^{-1},\log n\})$ where $T = \frac{R}{\delta}$. Along the lines of proof of Theorem 7.1.3, the additional $\frac{1}{\delta}$ factor is from $\frac{1}{\delta}$ parallel invocations of the constant-factor approximation algorithm. Therefore Theorem 8.1.3 will follow. In the subsequent sections, we prove Lemma 8.3.3 and Theorem 8.3.2.

8.3.1 Proof of Lemma 8.3.3

Lemma 8.3.3. *If $\lambda > 1 + 8\delta$ (otherwise the algorithm has stopped) and $\mathbf{x}^c \in \mathcal{Q}^c$ (will be an invariant) some $U \in \mathcal{O}_\delta(V')$ contains any $p_{ij,i}$ such that either (i) both $i, j \notin U$ or (ii) $x^c_{ip_{ij,i}} = x^c_{p_{ij,j}j} = 0$, then $\lambda_U < \lambda - \delta^2$. As a consequence of (i), $a_{ij} = \eta_{ip_{ij,i}} + \eta_{p_{ij,j}j} - \eta_{p_{ij,i}p_{ij,j}} \geq 0$. As a consequence of (ii), we can compute the z_U in time $O(m' \operatorname{poly}\{\delta^{-1}, \log n\})$ where $m' = |\{(i,j) | \textsc{Short}(\mathbf{x}^c)_{ij} \neq 0\}|$.*

Proof. Define $X^c_U = \sum_{(i',j'):i',j' \in U} x^c_{i'j'}$ for all sets U (even or odd). For an odd set U we have $\lambda_U \tilde{b^c}_U = X^c_U$.

We focus on the first part of the lemma. Assume (for contradiction) $\lambda_U \geq \lambda - \delta^2 \geq 1 + 7\delta$ for such a set U as in the statement of the lemma. Note $b^c_{p_{ij,i}} = c_{ij}$. Also note that for any $U \in \mathcal{O}_{\delta L}$ we have $(1-\delta)\left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor \leq \tilde{b^c}_U$ (based on $\mathcal{F}1$ from Fact 7.4.1 in page 152).

If $p_{ij,j} \notin U$ define $U' = U - \{p_{ij,i}\}$. In this case $X^c_U = X^c_{U'}$ (in both cases (i) and (ii)). If c_{ij} is odd then $||U'||_{b,c}$ is even but in that case:

$$\begin{aligned}\lambda_U \tilde{b^c}_U = X^c_U = X^c_{U'} &\leq \frac{1}{2} \sum_{i' \in U'} \sum_{j'} x^c_{i'j'} \leq \frac{1}{2}\lambda \sum_{i' \in U'} \tilde{b^c}_{i'} = \frac{1}{2}\lambda(1-4\delta)||U'||_{b,c} \\ &= \lambda(1-4\delta)\left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor \leq \lambda(1-4\delta)\frac{\tilde{b^c}_U}{(1-\delta)} \leq \lambda(1-\delta)\tilde{b^c}_U \quad (\text{since } \delta \leq \tfrac{1}{8})\end{aligned}$$

which implies $\lambda_U \leq (1-\delta)\lambda$. But if c_{ij} is even (and therefore ≥ 2) then U' cannot be a single element because then $0 = X^c_{U'} = X^c_U = \lambda_U \tilde{b^c}_U > (1+7\delta)\tilde{b^c}_U$ which is impossible. Therefore U' is an odd-set and $\lambda_{U'}, \tilde{b^c}_{U'}$ is defined. Now $||U'||_{b,c} = ||U||_{b,c} - c_{ij}$, and therefore $\tilde{b^c}_U - \tilde{b^c}_{U'} = \frac{c_{ij}}{2} - \delta^2 c_{ij}||U||_{b,c} \geq (1-2\delta)c_{ij}/2 \leq c_{ij}/4$ since $U \in \mathcal{O}_{\delta L}$ and

$\delta \leq 1/8$, therefore $||U||_{b,c} \leq \frac{1}{\delta}$. Now, $\lambda \tilde{b^c}_{U'} \geq \lambda_{U'} \tilde{b^c}_{U'} = X^c_{U'} = X^c_U = \lambda_U \tilde{b^c}_U$ implying:

$$\lambda \geq \lambda_U \frac{\tilde{b^c}_U}{\tilde{b^c}_{U'}} = \lambda_U \left(1 + \frac{c_{ij}}{4\tilde{b^c}_{U'}}\right) \geq \lambda_U \left(1 + \frac{\delta}{4}\right)$$

implying $\lambda(1 - \delta/8) \geq \lambda_U$. Therefore if $p_{ij,j} \notin U$ then $\lambda(1 - \delta^2) \geq \lambda_U$ in all cases. Therefore, for the first part all that remains to be considered is $p_{ij,j} \in U$.

If $p_{ij,j} \in U'$, define $U' = U - \{p_{ij,i}, p_{ij,j}\}$. Observe that $||U'||_{b,c} = ||U||_{b,c} - 2c_{ij}$ is always odd and $X^c_{U'} = X^c_U - c_{ij}$ (in either case of (i) $i, j \notin U$ or (ii) $x^c_{ip_{ij,i}} = x^c_{p_{ij,j}j} = 0$). Since $X^c_U > (1 + 7\delta)\tilde{b^c}_U$ and $\tilde{b^c}_U \geq (1 - \delta) \left\lfloor \frac{||U||_{b,c}}{2} \right\rfloor \geq (1 - \delta)c_{ij}$. Therefore $X^c_{U'} \geq (1+7\delta)(1-\delta)c_{ij} - c_{ij} > 0$. Therefore U' is not a singleton set and $\lambda_{U'}, \tilde{b^c}_{U'}$ is defined. But then, $\tilde{b^c}_U - \tilde{b^c}_{U'} = c_{ij} - 2\delta^2 c_{ij}||U||_{b,c} \geq (1-2\delta)c_{ij}$. Using $1+7\delta \leq 2, 14\delta \leq 2, c_{ij} \geq 1, \tilde{b^c}_{U'} \leq \frac{1}{\delta}$, we have:

$$\begin{aligned}
\lambda \tilde{b^c}_{U'} \geq \lambda_{U'} \tilde{b^c}_{U'} = X^c_{U'} = X^c_U - c_{ij} &= \lambda_U \left(\tilde{b^c}_U - \frac{c_{ij}}{\lambda_U}\right) \geq \lambda_U \left(\tilde{b^c}_U - \frac{c_{ij}}{1 + 7\delta}\right) \\
&\geq \lambda_U \tilde{b^c}_{U'} \left(\frac{\tilde{b^c}_U}{\tilde{b^c}_{U'}} - \frac{c_{ij}}{(1 + 7\delta)\tilde{b^c}_{U'}}\right) \geq \lambda_U \tilde{b^c}_{U'} \left(1 + \frac{(1 - 2\delta)c_{ij}}{\tilde{b^c}_{U'}} - \frac{c_{ij}}{(1 + 7\delta)\tilde{b^c}_{U'}}\right) \\
&= \lambda_U \tilde{b^c}_{U'} \left(1 + \frac{(5 - 14\delta)\delta c_{ij}}{(1 + 7\delta)\tilde{b^c}_{U'}}\right) = \lambda_U \tilde{b^c}_{U'} \left(1 + \frac{3\delta^2}{2}\right) \\
&\geq \lambda_U \tilde{b^c}_{U'} \left(\frac{1}{1 - \delta^2}\right) \quad \text{(Using } \delta \leq 1/8)
\end{aligned}$$

and therefore the first part of the lemma follows, that is, in either case of (i) $i, j \notin U$ or (ii) $y^c_{ip_{ij,i}} = y^c_{p_{ij,j}j} = 0$ we have $\lambda_U \leq \lambda - \delta^2$. Therefore no such U is chosen in L^c or is needed to be counted as a candidate in the number of odd sets in Theorems 7.2.1 and 7.2.2 in this context. For the second part of the lemma, observe that based on

Definition 8.3.1;

$$
\begin{aligned}
a_{ij} &= \left(x_i^c + x_{p_{ij,i}}^c + \sum_{U;i,p_{ij,i}\in U} z_U^c \right) + \left(x_j^c + x_{p_{ij,j}}^c + \sum_{U;j,p_{ij,j}\in U} z_U^c \right) \\
&\qquad - \left(x_{p_{ij,i}}^c + x_{p_{ij,j}}^c + \sum_{U;p_{ij,i},p_{ij,j}\in U} z_U^c \right) \\
&= x_i^c + x_j^c + \sum_{U;i,p_{ij,i}\in U} z_U^c + \sum_{U;j,p_{ij,j}\in U} z_U^c - \sum_{U;p_{ij,i},p_{ij,j}\in U} z_U^c
\end{aligned}
$$

but then a_{ij} can only be negative if there exists a set $U \in L^c$ such that $p_{ij,i}, p_{ij,j} \in U$ and $i, j \notin U$. The first part of the lemma, case (i), just ruled out that possibility. Case (ii) rules out computing odd sets containing $p_{ij,i}$ or $p_{ij,j}$ such that $y_{ip_{ij,i}}^c = 0$ or that $\textsc{Short}(\mathbf{x}^c)_{ij} = 0$. It is also worth noting that $x_{p_{ij,i}}$ is independent of $y_{ip_{ij,i}}^c$ for $\mathbf{x}^c \in \mathcal{Q}^c$ and can be stored implicitly. □

8.3.2 Proof Of Theorem 8.3.2

Theorem 8.3.2. *If $\beta \leq (1-4\delta)\beta^*$ a solution to LP34 always exists*

$$
\sum_{(i,j)\in E} a_{ij}\tilde{x}_{ij} \leq \gamma^c - K \qquad \textit{subject to } \tilde{\mathbf{x}} \in \begin{cases} \sum_{i,j} w_{ij}x_{ij} \geq \beta & \\ x_{ij} \leq c_{ij} & \forall (i,j) \in E \\ \sum_j x_{ij} \leq b_i & \forall i \in V \\ x_{ij} \geq 0 & \end{cases} \tag{LP34}
$$

where $K = \sum_{(i,j)\in E} \eta_{p_{ij,i}p_{ij,j}} c_{ij}$ as in Definition 8.3.1. Moreover, if $a_{ij} = \eta_{i,p_{ij,i}} + \eta_{p_{ij},j} - \eta_{p_{ij,i}p_{ij,j}} \geq 0$ for all $(i,j) \in E$, and LP34 is feasible then we can solve LP33

$$
\sum_{(i',j')} \tilde{x}_{i'j'}^c \eta_{i',j'} \leq \gamma^c \qquad \textit{subject to } \tilde{\mathbf{x}}^c \in \mathcal{Q}^c[\beta] \tag{LP33}
$$

in $O(\frac{m}{\delta}\ln(\frac{1}{\delta}))$ *time using* $O(\min\{m,B\}\cdot\frac{1}{\delta}\ln(\frac{1}{\delta}))$ *space and* $\mathfrak{q}=O(\ln(\frac{1}{\delta}))$ *passes over the list of edges of* G*. Moreover if we define* $S=\{(i,j)\in E|\textsc{Short}(\tilde{\mathbf{x}}^c)_{ij}>0\}$ *then* $\sum_{(i,j)\in S}w_{ij}\leq\mathfrak{q}\beta^*$ *and* $|S|\leq O(\min\{m,B\}\ln\frac{1}{\delta})$*. This also provides us with an initial solution with* $\lambda_0=O(\ln\frac{1}{\delta})$ *and* $\beta_0=O(\beta^*\ln\frac{1}{\delta})$*.*

Proof. Suppose $\beta\leq(1-4\delta)\beta^*$ then consider the optimum solution $\mathbf{x}^*=\{x^*_{ij}\}$ over G. Further consider $\mathbf{x}'=\mathbf{x}^*(1-4\delta)$. Note that $\sum_{(i,j)\in E}y'_{ij}w_{ij}\geq\beta$. Now $\textsc{Long}(\mathbf{x}')\in\mathcal{Q}^c$ satisfies LP32 with $\lambda=1$ (note that $\tilde{b^c}_U\geq(1-\delta)\left\lfloor\frac{||U||_{b,c}}{2}\right\rfloor\geq(1-4\delta)\left\lfloor\frac{||U||_{b,c}}{2}\right\rfloor$) and $\sum_{(i',j')}\textsc{Long}(\mathbf{x}')_{i'j'}w^c_{i'j'}=\sum_{(i,j)\in E}x'_{ij}w_{ij}\geq\beta$. Multiplying rows of $\mathbf{A}^c_L$ with the non-negative numbers $\{y^c_i\},\{z^c_U\}$ we get that

$$\sum_{(i',j')}\textsc{Long}(\mathbf{x}')_{i'j'}\left(y^c_{i'}+y^c_{j'}+\sum_{U\in\mathcal{L}^c;i',j'\in U}z^c_U\right)=\sum_{(i',j')}\textsc{Long}(\mathbf{x})_{i'j'}\eta_{i',j'}\leq\gamma^c$$

$$\iff\sum_{(i,j)\in E}a_{ij}x'_{ij}\leq\gamma^c-K$$

This would imply LP34 is feasible. This proves the first part of the theorem. If LP34 is feasible, then $a_{ij}\geq 0$ imply $\gamma^c-K\geq 0$ moreover for any $\varrho\geq 0$;

$$\sum_{(i,j)}(w_{ij}-\varrho a_{ij})x_{ij}\geq\beta-\varrho(\gamma^c-K)$$

$$\text{subject to }\mathbf{x}\begin{cases}x_{ij}\leq c_{ij} & \forall(i,j)\in E\\ \sum_j x_{ij}\leq b_i & \forall i\in V\\ x_{ij}\geq 0\end{cases}\tag{LP35}$$

the system LP35 is feasible. Suppose for any $\varrho \geq 0$, we can find $\mathbf{x}$ such that:

$$\sum_{(i,j)} (w_{ij} - \varrho a_{ij}) x_{ij} \geq \left(1 - \frac{\delta}{2}\right) (\beta - \varrho(\gamma^c - K))$$

$$\text{s. t. } \mathbf{x} \in \mathcal{Q} : \begin{cases} x_{ij} \leq c_{ij} & \forall (i,j) \in E \\ \sum_j x_{ij} \leq \lambda_0 b_i / 2 & \forall i \in V \\ x_{ij} \geq 0 \end{cases} \tag{LP36}$$

We apply Lemma 6.2.3, proved in page 116 (using $\beta_2 = \gamma^c - K \geq 0, \beta_1 = \beta \geq 0, q = 1 - \delta/2, r = \delta/2$ and that $\sum_{i,j} a_{ij} x_{ij} \geq 0$ for $\mathbf{x} \in \mathcal{Q}$ since all a_{ij} are non-negative) and we will get $\mathbf{x}^\dagger$ satisfying:

$$\begin{cases} \sum_{i,j} a_{ij} x^\dagger_{ij} \leq (\gamma^c - K) \\ \sum_{i,j} w_{ij} x^\dagger_{ij} \geq (1-r) q \beta = \left(1 - \frac{\delta}{2}\right)\left(1 - \frac{\delta}{2}\right) \beta \geq (1-\delta)\beta \end{cases}$$

and (C1)

$$\begin{cases} x^\dagger_{ij} \leq c_{ij} & \forall (i,j) \in E \\ \sum_j x^\dagger_{ij} \leq \lambda_0 b_i / 2 & \forall i \in V \\ x^\dagger_{ij} \geq 0 \end{cases}$$

Now consider $\textsc{Long}(\mathbf{x}^\dagger)$. Based on the condition (C1), $\textsc{Long}(\mathbf{x}^\dagger) \in \mathcal{Q}^c[\beta]$ and

$$\sum_{(i,j)} a_{ij} x^\dagger_{ij} \leq (\gamma^c - K) \qquad \Longrightarrow \qquad \sum_{(i',j')} \textsc{Long}(\mathbf{x}^\dagger)_{i'j'} \left(y^c_{i'} + y^c_{j'} + \sum_{U \in \mathcal{L}^c; i', j' \in U} z^c_U \right) \leq \gamma^c$$

by rearrangement and we have a solution for LP33. Note that $|\{(i',j') | \textsc{Long}(\mathbf{x}^\dagger)_{i'j'} > 0\}|$ is at most twice the maximum number of non-zero entries in any $\mathbf{x}$ returned to satisfy LP36. In what follows we show that this number is at most $O(\min\{B, m\})$, and $\lambda_0/2 = 7 \ln \frac{2}{\delta}$ and we find a solution to LP36 in $O(\lambda_0 m)$ time, completing the proof of Theorem 8.3.2.

We rewrite LP35 as a maximization problem LP37 using $w^e_{ij} = \max\{0, w_{ij} - \varrho a_{ij}\}$ and $c'_{ij} = \min\{c_{ij}, b_i, b_j\}$. We also formulate the (slightly modified) dual LP38.

$$\begin{aligned}
&\tau^* = \max \textstyle\sum_{i,j} w^e_{ij} x_{ij} \\
&x_{ij} \le c'_{ij} && \forall (i,j) \in E \\
&\textstyle\sum_j x_{ij} \le b_i && \forall i \in V \\
&x_{ij} \ge 0
\end{aligned} \tag{LP37}$$

$$\begin{aligned}
\tau^* = \quad & \min \textstyle\sum_i p_i + \sum_{i,j} q_{ij} \\
& \tfrac{p_i}{b_i} + \tfrac{p_j}{b_j} + \tfrac{q_{ij}}{c'_{ij}} \le w^e_{ij} \quad \forall (i,j) \in E \\
& p_i, q_{ij} \ge 0
\end{aligned} \tag{LP38}$$

If LP35 is feasible then the optimum solution $\hat{x}_{ij}$ of LP37 satisfies $\sum_{i,j} w^e_{ij} \hat{x}_{ij} = \tau^* \ge \beta - \varrho(\gamma^c - K)$. Note that $\hat{x}_{ij}$ can be fractional (and our proof of feasibility of LP34 used a fractional vector).

Now suppose for any $b_i, c_{ij} \ge 0$ we can find a solution $\mathbf{x}^{(1)}$ such that $\sum_{i,j} w^e_{ij} x^{(1)}_{ij} = \tau_1 \ge \tau^*/7$ using Theorem 7.6.2 in Section 7.6. Define $\tau^*(1) = \tau^*$. We run an iterative procedure where we set the constraints of LP37 to $x_{ij} \le \max\{0, c'_{ij} - x^{(1)}_{ij}\}$ then the optimum solution of this modified LP37 denoted by $\tau^*(2)$ is at least $\tau^*(1) - \tau_1$. Otherwise it is easy to see that the optimum solution of unmodified LP37 cannot be τ^*. Therefore we can obtain a solution $\mathbf{x}^{(2)}$ such that $\sum_{i,j} w^e_{ij} x^{(2)}_{ij} = \tau_2 \ge \tau^*(2)$. We now repeat the process by modifying LP37 to $x_{ij} \le \max\{0, c'_{ij} - x^{(1)}_{ij} - x^{(2)}_{ij}\}$. Proceeding in this fashion we obtain solutions $\left\{x^{(\ell)}_{ij}\right\}_{\ell=1}^{7\ln\frac{2}{\delta}}$. *Observe, that by construction* $\sum_{\ell=1}^{7\ln\frac{2}{\delta}} x^{(\ell)}_{ij} \le c_{ij}$ *for all* (i,j) *and therefore the union of these* $7\ln\frac{2}{\delta}$ *solutions satisfies Invariant 1.* We now claim:

$$\sum_{\ell=1}^{r} \sum_{i,j} w^e_{ij} x^{(\ell)}_{ij} \ge \left(1 - \left(\frac{6}{7}\right)^r\right) \tau^* \tag{8.3.1}$$

We prove Equation (8.3.1) by induction for all τ^* for a fixed upper bound on r; we just proved the base case for $r = 1$ since $\tau_1 \geq \tau^*/7$. In the inductive case, applying the hypothesis on $2, \ldots, r$ we get $\sum_{\ell=2}^{r} \sum_{i,j} w_{ij}^e x_{ij}^{(\ell)} \geq \left(1 - \left(\frac{6}{7}\right)^{r-1}\right) \tau^*(2)$. Now if $\tau^*(2) \geq \frac{6}{7}\tau^*$ then

$$\sum_{\ell=1}^{r} \sum_{i,j} w_{ij}^e x_{ij}^{(\ell)} \geq \frac{1}{7}\tau^* + \left(1 - \left(\frac{6}{7}\right)^{r-1}\right) \frac{6}{7}\tau^*$$

and Equation (8.3.1) holds. Otherwise if $\tau^*(2), 6\tau^*/7$ then:

$$\begin{aligned}\sum_{\ell=1}^{r} \sum_{i,j} w_{ij}^e x_{ij}^{(\ell)} &\geq (\tau^* - \tau^*(2)) + \left(1 - \left(\frac{6}{7}\right)^{r-1}\right) \tau^*(2) \\ &\geq \tau^* - \left(\frac{6}{7}\right)^{r-1} \tau^*(2) \geq \left(1 - \left(\frac{6}{7}\right)^{r}\right) \tau^*\end{aligned}$$

Therefore in all cases we will have $\sum_{\ell=1}^{7 \ln \frac{2}{\delta}} w_{ij}^e x_{ij}^{(\ell)} \geq \left(1 - \frac{\delta}{2}\right) \tau^*$. This implies that if we collect $7 \ln \frac{2}{\delta}$ solutions given by the 7 approximation algorithm then their union will be more than $\left(1 - \frac{\delta}{2}\right) \tau^* \geq \left(1 - \frac{\delta}{2}\right) (\beta - \varrho(\gamma^c - K))$ provided LP35 is feasible.

Therefore if for any $\varrho \geq 0$ we collect these many solutions and find that $\sum_{\ell=1}^{7 \ln \frac{2}{\delta}} w_{ij}^e x_{ij}^{(\ell)} < (1 - \delta/2)(\beta - \varrho(\gamma^c - K))$ then we know that it is impossible that $\beta \leq (1 - 4\delta)\beta^*$ and we reduce β. Note that the number of times we reduce β is now $\frac{1}{\delta} \ln \ln \frac{1}{\delta}$, and we may spend $O(\ln \frac{1}{\delta})$ passes for each such reduction. This can still be ignored in comparison to the number of passes $O(\lambda_0 \delta^{-4} (\ln^2 \frac{1}{\delta}) \ln n)$.

On the other hand, if the union of $7 \ln \frac{2}{\delta}$ solutions is at least $(1-\delta/2)(\beta-\varrho(\gamma^c-K))$ (for each different ϱ) then we have a solution for LP36; and by the previous discussion (applying Lemma 6.2.3, which implies a repeating for $O(\ln \frac{1}{\delta})$ non-negative values of ϱ) we have a solution as desired by the Theorem. This proves the bound on $\mathfrak{q} = O(\ln^2 \frac{1}{\delta})$.

For the initial solution, set $\varrho = 0$. This proves the bound on λ_0, β_0 and the

number of edges. Finally note that in each pass of the algorithm we compute a feasible, integral, capacitated b–Matching (with capacities less than that of the original graph) and therefore $\sum_{(i,j)\in S} w_{ij} \leq \mathfrak{q}\beta^*$. $\square$

8.4 Rounding Fractional Capacitated b-Matching

In this section, we prove the following:

Theorem 8.1.4. *Given a fractional capacitated b-matching* $\mathbf{x}$ *such that the optimum solution is at most* β^* *and* $\sum_{(i,j)\in\hat{E}} w_{ij} \leq R\beta^*$ *where* $\hat{E} = \{(i,j)|x_{ij} > 0\}$, *we can find an integral b-matching of weight at least* $(1-\delta)\sum_{i,j} w_{ij}y_{ij} - \delta\beta^*$ *in* $O(m'R\delta^{-3}\log(R/\delta))$ *time and* $O(m'/\delta^2)$ *space where* $m' = |\hat{E}|$. *Note that if the fractional solution is at least* $(1-14\delta)$*-approximate then we have a* $(1-16\delta)$*-approximate integral solution.*

The rounding is achieved by Algorithm 31. The general outline of the algorithm and its proof is similar to the uncapacitated case discussed in Section 7.8.

Lemma 8.4.1. $x_{ij}^{(1)}$ *is a feasible fractional capacitated b–Matching in* $G_c^{(1)}$.

Proof. Consider $\textsc{Long}(\mathbf{x}^{(1)})$ and $\textsc{Long}(G)$. The only vertices whose capacities were affected in $\textsc{Long}(G)$ are the following vertices: (i) the corresponding vertex in G has an edge incident to it in $\mathcal{M}_c^{(1)}$ and (ii) the corresponding edge $(i,j) \in G$ had $c_{ij} > \lceil x_{ij}^{(1)} \rceil + 1$.

Algorithm 31 Rounding a fractional capacitated b–Matching in small space (Part I)

1: First Phase: **Removing edges with large multiplicities** (no change from Algorithm 30 except tracking edge capacities). $t = \lceil 2/\delta \rceil$.

(a) If $x_{ij} \geq t$ add $\hat{x}_{ij}^{(0)} = \lfloor x_{ij} \rfloor - 1$ copies of (i,j) to $\mathcal{M}_c^{(0)}$.

(b) Set $x_{ij}^{(1)} = \begin{cases} 0 & \text{if } x_{ij} \geq t \\ x_{ij} & \text{otherwise} \end{cases}$. Set $b_i^{(1)} = \min\left\{b_i - \sum_j \hat{x}_{ij}^{(0)}, \lceil \sum_j x_{ij}^{(1)} \rceil + 1\right\}$ and $c_{ij}^{(1)} = \min\{c_{ij}, \lceil x_{ij}^{(1)} \rceil + 1\}$. This describes the graph $G_c^{(1)}$. Note $c_{ij}^{(1)} \leq t + 1$.

2: Second Phase: **Subdividing vertices with large multiplicities.** (no change from Algorithm 30 except tracking edge capacities). We set $c_{i'j'}^{(2)} = c_{ij}^{(1)}$ where the edge (i,j) got assigned to i' and j' which are copies of i and j respectively. This defines $G_c^{(2)}$. Note *the edges are not split*, only vertices are split.

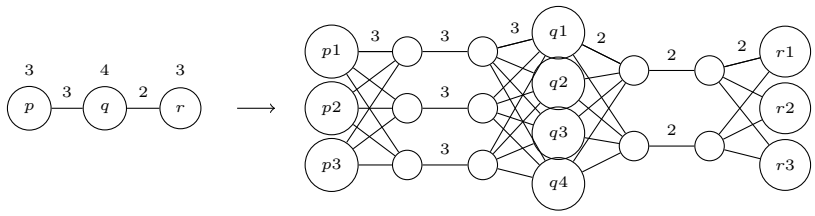


Figure 8.1: Example of Reduction from the maximum capacitated b–Matching Problem to the Maximum Matching Problem. The edges in the complete bipartite graphs have the same weight.

Algorithm 31 Rounding a fractional capacitated b–Matching in small space (Part II)

1: Third Phase: **Reducing the problem to a weighted matching on small graph.** (different from Algorithm 30). The following steps can be viewed as first constructing $\textsc{Long}(G_c^{(2)})$ and then applying the third phase of Algorithm 30. In more details:

(a) For each $i \in V^{(2)}$ with $b_i^{(2)}$, create $i(1), i(2), \cdots, i(b_i^{(2)})$. For each edge $e = (i, j)$, we create $2c_{ij}^{(2)}$ vertices $p_{ei,1}, p_{ei,2}, \cdots, p_{ei,c_{ij}^{(2)}}, p_{ej,1}, p_{ej,2}, \cdots, p_{ej,c_{ij}^{(2)}}$.

(b) Add edges $(p_{ei,\ell}, p_{ej,\ell})$ with edge weight w_{ij}. Add a complete bipartite graph between $i_1, i_2, \cdots$ and $p_{ei,1}, p_{ei,2}, \cdots$ with edge weight w_{ij}. See Figure 8.1.

(c) Run any fast approximation for finding a $(1 - \frac{\delta}{4(R+1)})$-approximate maximum weighted matching in $G^{(3)}$ let this matching be $\mathcal{M}_c^{(3)}$.

2: Final Phase: **Output** (same as Algorithm 30). Matching $\mathcal{M}_c^{(3)}$ implies a b–Matching $\mathcal{M}_c^{(2)}$ in $G_c^{(2)}$ of same weight (merge edges) which in turn implies a b–Matching $\mathcal{M}_c^{(1)}$ in $G_c^{(1)}$ of same weight (merge vertices). $\mathcal{M}_c^{(0)} \cup \mathcal{M}_c^{(1)}$ is the final solution.

In both cases the difference between the sum of the new edge multiplicities and the new capacities (the slack) is at least 1 and the first part of Lemma 7.8.1 tells us that these vertices in $\textsc{Long}(G)$ cannot be part of a violated odd-set in $\textsc{Long}(G)$. Therefore $\textsc{Long}(\mathbf{x}^{(\mathbf{1})})$ is feasible for $\textsc{Long}(G)$. The lemma follows from Theorem 8.2.3. $\square$

Therefore the only task that remains is to find a $(1-\delta)$ approximate rounding of the fractional solution $x_{ij}^{(1)}$ on $G_c^{(1)} = (V, E^{(1)})$ with vertex and capacities $\{b_{ij}^{(1)}\}$ and $\{c_{ij}^{(1)}\}$ respectively.

Lemma 8.4.2. *Let $OPT^{(1)}$ be the maximum capacitated b–Matching of $G_c^{(1)} = (V, E^{(1)})$. Let $\boldsymbol{\mathcal{W}} = \sum_{(i,j)\in E^{(1)}} c_{ij}^{(1)} w_{ij}$. Then $\boldsymbol{\mathcal{W}} \leq 2R\beta^* + OPT^{(1)} \leq (2R+1)\beta^*$.*

Proof. $OPT^{(1)} \leq \beta^*$ because we are only decreasing vertex and edge capacities. By construction, $c_{ij}^{(1)} \leq \lceil x_{ij}^{(1)} \rceil + 1 \leq x_{ij}^{(1)} + 2$. Note $\sum_{(i,j)\in E^{(1)}} x_{ij}^{(1)} w_{ij} \leq OPT^{(1)}$. We have already seen that $\sum_{(i,j)\in \hat{E}} w_{ij} \leq R\beta^*$. Observe that $\sum_{(i,j)\in E^{(1)}} w_{ij} \leq \sum_{(i,j)\in \hat{E}} w_{ij}$. The lemma follows. $\square$

Lemma 8.4.3. *Algorithm 31 outputs a capacitated b–Matching of weight at least $(1-\delta)\sum_{(i,j)\in E} w_{ij} x_{ij} - \delta\beta^*$.*

Proof. Since second phase is exactly the same as in the uncapacitated case in Section 7.8, we have $\sum_{(i,j)\in E^{(2)}} w_{ij} x_{ij}^{(2)} \geq (1-\delta)\sum_{(i,j)\in E^{(1)}} w_{ij} x_{ij}^{(1)}$. Moreover since we did not introduce any new edge $\boldsymbol{\mathcal{W}} = \sum_{(i,j)\in E^{(2)}} w_{ij} c_{ij}^{(2)} = \sum_{(i,j)\in E^{(1)}} c_{ij}^{(1)} w_{ij} \leq (2R+1)\beta^*$.

Now consider the step 4 (third phase) of Algorithm 31; we first construct $\textsc{Long}(G_c^{(2)})$ where the edge (i,j) gets subdivided to $(i, p_{ij,i}), (p_{ij,i}, p_{ij,j}), (p_{ij,j}, j)$ with

weights w_{ij}, w_{ij}, w_{ij}. Based on [86, Theorem 32.4, Vol A, page 567], there exists a feasible uncapacitated b–Matching in $\textsc{Long}(G_c^{(2)})$ whose weight is at least $\sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)} + \boldsymbol{\mathcal{W}}$. Moreover if we find an uncapacitated b–Matching in $\textsc{Long}(G_c^{(2)})$ of weight W then there exists a capacitated b–Matching in $G_c^{(2)}$ of weight $W - \boldsymbol{\mathcal{W}}$.

The step 4 of Algorithm 31 then reduces finding an uncapacitated b–Matching in $\textsc{Long}(G_c^{(2)})$ to a matching in $G_c^{(3)}$ where we replace every edge by a complete bipartite graph corresponding to the capacities of the two endpoints. Let the weight of the maximum matching of this graph $G_c^{(3)}$ be $w(\mathcal{M}*)$.Then $w(\mathcal{M}*) \geq \sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)} + \boldsymbol{\mathcal{W}}$. Suppose that we find a $\frac{\delta}{4T+4}$-approximate maximum matching in $G_c^{(3)}$, using the algorithm in [34, 35] which takes time $|E(G_c^{(3)})|$ times
$O(\frac{R}{\delta}\log(R/\delta))$ which is $O(m'R\delta^{-3}\log(R/\delta))$. This gives us a matching of weight at least $w(\mathcal{M}^*) - \delta w(\mathcal{M}*)$ which corresponds to a capacitated b–Matching in $G_c^{(2)}$ with weight $w(\mathcal{M}^*) - \delta w(\mathcal{M}*) - \boldsymbol{\mathcal{W}}$. Now

$$w(\mathcal{M}^*) - \delta w(\mathcal{M}*) - \boldsymbol{\mathcal{W}} \geq \sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)} + \boldsymbol{\mathcal{W}} - \delta w(\mathcal{M}*) - \boldsymbol{\mathcal{W}}$$

$$= \sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)} - \frac{\delta w(\mathcal{M}^*)}{4(R+1)} \geq \sum_{(i,j)\in E^{(2)}} w_{ij}x_{ij}^{(2)} - \delta\beta^* \geq (1-\delta)\sum_{(i,j)\in E^{(1)}} w_{ij}x_{ij}^{(1)} - \delta\beta^*$$

We get a matching $\mathcal{M}_c^{(1)}$ in $G_c^{(1)}$ of weight $w(\mathcal{M}_c^{(1)}) \geq (1-\delta)\sum_{(i,j)\in E^{(1)}} w_{ij}x_{ij}^{(1)} - \delta\beta^*$. Observe that $w(\mathcal{M}_c^{(0)}) \geq (1-\delta)\sum_{(i,j)\in E} w_{ij}\left(x_{ij} - x_{ij}^{(1)}\right)$ where $w(\mathcal{M}_c^{(0)}) = \sum_{(i,j)\in E} \hat{x}_{ij}^{(0)} w_{ij}$. Then $w(\mathcal{M}_c^{(0)}) + w(\mathcal{M}_c^{(1)})$ is at least $(1-\delta)\sum_{(i,j)\in E} w_{ij}x_{ij} - \delta\beta^*$ as desired. This proves Lemma 8.4.3, which in turn proves Theorem 8.1.4. $\square$

Chapter 9

Application III: Dual-Primal Approach to Nonbipartite b-Matching

Chapter Outline: In this chapter, we apply the dual-primal approach in Section 6.3 to the nonbipartite b–Matching problem. We presented the dual-primal approach in Section 6.3 with an example of the bipartite MWM. The algorithm for the nonbipartite b–Matching problem is similar to the bipartite MWM. However, we generalize its building blocks in order to handle odd set constraints. Recall that the standard

LP for the nonbipartite b–Matching problem:

$$\begin{aligned} \beta^* = \text{LP22}(\mathbf{b}) = \quad & \max \textstyle\sum_{(i,j)} w_{ij} x_{ij} \\ & \textstyle\sum_j x_{ij} \leq b_i && \forall i \in V \\ & \textstyle\sum_{(i,j):i,j \in U} x_{ij} \leq \lfloor ||U||_b/2 \rfloor && \forall U \in \mathcal{O} \\ & x_{ij} \geq 0 \end{aligned} \tag{LP22}$$

We give the generalized version of the dual-primal approach and building blocks in Section 9.1. In Sections 9.2 and 9.3, we present algorithms for the unweighted and weighted b–Matching problems. In Section 9.4, we present the deferred sparsification algorithm which is a generalization of the deferred degree counting (see Section 6.3.2). In Section 9.5, we present a rounding algorithm for the compressed fractional solution.

9.1 A Framework for a Dual-Primal Approach

Suppose we have the following packing optimization problem:

$$\begin{aligned} \beta^* = & \max \mathbf{c}^{\mathrm{T}} \mathbf{x} \\ & \mathcal{A}^{\mathrm{T}} \mathbf{x} \leq \mathcal{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{Primal}$$

where $\mathbf{c} \in \mathbb{R}_+^m$ (where $\mathbb{R}_+$ denotes the nonnegative reals) $\mathcal{b} \in \mathbb{R}_+^M$ and $\mathcal{A} \in \mathbb{R}_+^{m \times M}$. In the example of the b–Matching problem, $\mathbf{x}$ is a fractional assignment over edges in the graph and so $m = \text{poly}(n)$ for the problem size n, for example, the number of vertices.

Definition 9.1.1. Define PRIMAL to be **Dual-Primal amenable** if for the feasibility problem DUAL(β, Λ) defined as

$$\text{Outer Covering: } \{\mathbf{A}\mathbf{y} \geq \mathbf{c}$$

$$\mathbf{Q}(\beta,\Lambda): \begin{cases} \text{Inner Packing:} & \{\mathbf{P}_o\mathbf{y} \leq \Lambda\mathbf{q}_o \\ \tilde{\mathbf{Q}}(\beta): & \begin{cases} \mathbf{b}^{\mathrm{T}}\mathbf{y} \leq \beta \\ \mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i \\ \mathbf{y} \geq \mathbf{0} \end{cases} \end{cases} \qquad (\text{DUAL}(\beta,\Lambda))$$

where $\mathbf{A} \in \mathbb{R}_+^{m\times N}$, $\mathbf{y}, \mathbf{b} \in \mathbb{R}_+^N$, $\mathbf{P}_o, \mathbf{P}_i \in \mathbb{R}_+^{n\times N}$, $\mathbf{q}_o, \mathbf{q}_i \in \mathbb{R}_+^n$, $\rho_o, \rho_i, \Lambda \in \mathbb{R}_+$ and the following holds:

(d1) $\mathbf{P}_o\mathbf{y} \leq \mathbf{q}_o$ and $\mathbf{y} \geq \mathbf{0}$ implies $\mathbf{A}\mathbf{y} \leq \rho_o\mathbf{c}$.

(d2) $\mathbf{P}_o\mathbf{y} \leq 2\mathbf{q}_o$ and $\mathbf{y} \geq \mathbf{0}$ implies $\mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i$.

(d3) $\mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i$ and $\mathbf{y} \geq \mathbf{0}$ implies $\mathbf{P}_o\mathbf{y} \leq \rho_i\mathbf{q}_o$.

(d4) If there exists $\mathbf{y}$ such that $\mathbf{A}\mathbf{y} \geq (1-3\epsilon)\mathbf{c}$ and $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq \beta$; then $\beta^* < \beta/(1-a_0\epsilon)$.

Of course the pairs $(\mathcal{A}, \mathbf{A})$ and $(\mathcal{b}, \mathbf{b})$ have to be related – in this paper $\mathbf{A}, \mathbf{b}$ will correspond to projections of $\mathcal{A}, \mathcal{b}$ to fewer rows and indices respectively (from M to N); that is, by padding $\mathbf{y}$ with 0 entries we can get $\mathbf{y} \in \mathbb{R}_+^M$ such that $\mathbf{A}\mathbf{y} \geq c$ and $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq \beta$ implies $\mathcal{A}\mathbf{y} \geq c$ and $\mathcal{b}^{\mathrm{T}}\mathbf{y} \leq \beta$. The transformation allows us to reduce M to N; in the example of the b–Matching problem, we will have $M = 2^n$ and $N = n^{O(1/\epsilon)}$. Definition 9.1.1 is stating that the addition of a small number of constraints $\mathbf{P}_o\mathbf{y} \leq \mathbf{q}_o$

does not affect the covering program but allow us to prove a width of ρ_o. Likewise there exists a relaxed version of $\mathbf{P}_o\mathbf{y} \leq \mathbf{q}_o$ which is $\mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i$, where the relaxation is by a factor ρ_i.

A Basic Algorithm: Suppose a_0, a_1, a_2 are small constants. Consider:

$$\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{y} \geq \left(1 - \frac{\epsilon}{2}\right)\mathbf{u}^{\mathrm{T}}\mathbf{c} \quad \text{subject to} \quad \mathbf{y} \in \mathbf{Q}(\beta, 2) \tag{Outer}$$

Suppose that for any nonnegative $\mathbf{u} \in \mathbb{R}_+^m$ we have an Oracle that either (i) provides a solution for Outer or (ii) provides a (compressed) feasible solution for Primal satisfying $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1 - a_1\epsilon)\beta$ along with additional properties $\mathcal{P}rop$. Moreover if Dual$(\beta, 1)$ is feasible it always returns the solution as (i). Also suppose that we have an initial solution $\mathbf{y}_0$ which satisfies $\mathbf{Q}(\beta_0, 1)$ along with $\mathbf{A}\mathbf{y}_0 \geq (1 - \epsilon_0)\mathbf{c}$ and $\beta^*/a_2 \leq \beta_0 < \beta^*/2$.

Then we can apply the fractional covering framework to Dual$(\beta, 2)$ (see Section 2.3.2 for the details of the framework). In cases (i), we can use the solution for Outer as a witness for the fractional covering framework. In cases (ii) we simply increase β by a factor of $(1 + \epsilon)$. Clearly we cannot increase β beyond $\beta^*/(1 - a_1\epsilon) < 2\beta^*$ due to case (ii) since we exceed the maximum possible value of Primal. Moreover, any admissible witness returned for case (i) for a particular value of β, continues to be admissible for larger values of β (see Section 6.1.2). Therefore eventually, we must have $O(\rho_o\epsilon^{-2}\log n)$ admissible witnesses for some β. Since the width is ρ_o using (d1) we have a $\mathbf{A}\mathbf{y} \geq (1 - 3\epsilon)\mathbf{c}$. Therefore from (d4) we have $\beta^* \leq \beta/(1 - a_0\epsilon)$. But if we remembered the $\mathbf{x}$ returned for $\beta/(1 + \epsilon)$ then we know $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1 - a_2\epsilon)\beta/(1 + \epsilon)$.

This implies that $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1-(a_0+a_1+1)\epsilon)\beta^*$; and the property $\mathcal{P}rop$ also holds. The total number of queries to ORACLE is $O(\rho_o(\epsilon^{-2}+\frac{1}{\epsilon}\log\frac{1}{1-\epsilon_0})\log m+\frac{1}{\epsilon}\log a_2)$. Assuming $\epsilon_0 < \frac{15}{16}$, i.e., sufficiently separated from 1, the number of queries to the oracle is $O(\rho_o\epsilon^{-2}\log m)$. After each query, any specific u_ℓ increases or decreases by at most $e^{\pm\epsilon}$; thus the entries of $\mathbf{u}$ are in range $m^{\pm O(1/\epsilon)}$.

Note that $\mathbf{Q}(\beta,2)$ is described using $O(n)$ constraints – and it is *plausible* that we can find a solution $\mathbf{y}$ to OUTER in each step which has $O(n\,\mathrm{poly}(\epsilon^{-1},\log n))$ nonzero entries. Then the $\mathbf{y}$ maintained by the overall covering formulation will also have $O(n\,\mathrm{poly}(\epsilon^{-1},\log n))$ non-zero entries provided $\log m = \mathrm{poly}(\epsilon^{-1},\log n)$. Note that the entries of the vector $\mathbf{u}$ are generated implicitly (using exponentials) by the fractional covering paradigm, namely $\mathbf{u}_\ell = \frac{1}{b_\ell}\exp\left(-\frac{(\mathbf{A}\mathbf{y})_\ell}{b_\ell}\right)$, then if we can compute $(\mathbf{A}\mathbf{y})_\ell$ then we would not even have to store $\mathbf{A}$ – if $n\,\mathrm{poly}(\epsilon^{-1},\log n) = o(m)$ we can solve the problem PRIMAL with $o(m)$ storage. *Observe that we get a solution for the system $\mathcal{A}^{\mathrm{T}}\mathbf{x} \geq \mathcal{b}$ in space which is sublinear!* Summarizing:

Theorem 9.1.2 (The basic algorithm)**.** *Assume* $\log m = \mathrm{poly}(\epsilon^{-1},\log n)$. *Suppose that given* $\mathbf{y}$ *with* $O(n\,\mathrm{poly}(\epsilon^{-1},\log n))$ *non-zero entries we can construct the vector* $\mathbf{u}$ *in* $O(1)$ *passes in the semi-streaming model. Suppose that for any nonnegative* $\mathbf{u} \in \mathbb{R}_+^m$ *we have an* ORACLE *that either (i) provides a solution for* OUTER *using* $O(n\,\mathrm{poly}(\epsilon^{-1},\log n))$ *space,* $O(m\,\mathrm{poly}(\epsilon^{-1},\log n))$ *time and* $O(1)$ *passes; (ii) provides a (compressed) feasible solution for* PRIMAL *satisfying* $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1-a_1\epsilon)\beta$ *along with additional properties* $\mathcal{P}rop$. *Moreover if* DUAL$(\beta,1)$ *is feasible it always returns the*

solution as (i). Then starting from a suitable initial solution we can find a compressed solution of PRIMAL *using* $O(n\,\mathrm{poly}(\epsilon^{-1}, \log n))$ *space and* $O(m\,\mathrm{poly}(\epsilon^{-1}, \log n))$ *time and* $O(\rho_o \epsilon^{-2} \log m)$ *passes over an implicit representation of* $\mathcal{A}^{\mathrm{T}}\mathbf{x} \geq \mathcal{b}$.

Improving the basic algorithm: Is the number of passes or rounds in Theorem 9.1.2 optimal? This is the central question we ask. We show that if we can provide suitably defined deferred data structures to store $\mathbf{u}$ (approximately) then we may not even need to compute all the entries of $\mathbf{u}$.

Theorem 9.1.3. *Suppose we have a* MICROORACLE *that either (i) provides a solution for* INNER *defined as:*

$$\tilde{\mathbf{Q}}(\beta) : \begin{cases} \mathbf{z}^{\mathrm{T}}\mathbf{P}_o\mathbf{y} \leq \dfrac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \\ \mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} \geq \left(1 - \dfrac{\epsilon}{8}\right)\mathbf{u}_s^{\mathrm{T}}\mathbf{c} \text{ and } \mathbb{G} \\ \mathbf{b}^{\mathrm{T}}\mathbf{y} \leq \beta \\ \mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i \\ \mathbf{y} \geq \mathbf{0} \end{cases} \tag{INNER}$$

or (ii) provides a solution of PRIMAL *with the additional property* $\mathcal{Prop}$. *Then we can find a solution of* PRIMAL *with property* $\mathcal{Prop}$ *using* $O(\frac{p\rho_o}{\epsilon}\frac{\log m}{\log n})$ *rounds of construction of deferred u–Sparsifiers, with* $O(\rho_o\epsilon^{-2}\log m)$ *in total, and* $O(\rho_o\rho_i\epsilon^{-2}(\log n)(\log m)\log\frac{1}{\epsilon})$ *queries to* MICROORACLE *with* $\varrho \leq \frac{12\mathbf{u}_s^{\mathrm{T}}\mathbf{c}}{13\mathbf{z}^{\mathrm{T}}\mathbf{q}_o}$. *The overall space complexity of the algorithm will be* $O(n^{1+1/p})$ *assuming* $p = O\left(\frac{\log n}{\log\frac{1}{\epsilon} + \log\log n}\right)$ *is not too large.*

Definition 9.1.4. (A Deferred u-Sparsifier.) Suppose given $\{\varsigma_\ell\}$ and the promise that $\varsigma_\ell/\chi \leq \mathbf{u}_\ell \leq \chi\varsigma_\ell$, we can construct a data structure that samples a subset of $\mathbf{u}$

and stores them. After the subset has been stored, the exact values of those store entries of $\mathbf{u}$ are revealed and the data structure constructs nonnegative $\mathbf{u}_s$ such that for some property $\mathbb{G}$ which is convex;

$$\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} \geq \left(1-\frac{\epsilon}{8}\right)\mathbf{u}_s^{\mathrm{T}}\mathbf{c} \text{ and } \mathbb{G} \quad \Longrightarrow \quad \mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{y} \geq \left(1-\frac{\epsilon}{2}\right)\mathbf{u}^{\mathrm{T}}\mathbf{c} \tag{Simple}$$

Assume $\mathbf{0}$ satisfies $\mathbb{G}$; further assume that there exists an algorithm to construct such deferred $\mathbf{u}$-sparsifiers using small space; say $O(n\,\mathrm{poly}(\chi,\epsilon,\log n))$ – in our case the running time will be $O(\frac{n\chi^2}{\xi^2}\log^4 n)$.

Lemma 9.1.5. *Suppose we are given an algorithm to construct deferred* $\mathbf{u}$*-Sparsifiers, an initial solution as described in the basic algorithm and a* ModOracle *that that either (i) provides a solution for* Sparse *given as:*

$$\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} \geq \left(1-\frac{\epsilon}{8}\right)\mathbf{u}_s^{\mathrm{T}}\mathbf{c} \quad \textit{and } \mathbb{G} \textit{ subject to} \quad \mathbf{y} \in \mathbf{Q}(\beta, 2) \tag{Sparse}$$

or (ii) provides a (compressed) feasible solution for Primal *satisfying* $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1 - a_1\epsilon)\beta$ *along with additional properties* $\mathcal{P}rop$*. Then we can find a* $(1-(a_0+a_1+1))$*-approximate solution with additional properties* $\mathcal{P}rop$ *using* $O(\frac{p\rho_o}{\epsilon}\frac{\log m}{\log n})$ *rounds of creating these data structures.*

Proof. Then we can simultaneously create $t = O(\log_{1+\epsilon}\chi)$ different data structures and simulate t steps of the basic algorithm without reading any further input. If $\chi = n^{1/(4p)}$ for some $p > 1$ then this reduces the effective number of steps (when we construct these data structures) by a factor of $O(\frac{1}{p\epsilon}\log n)$. □

Lemma 9.1.6. *Suppose we have* MicroOracle *that either (i) provides us a solution for* LagInner *defined as:*

$$\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} - \varrho\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\mathbf{y} \geq \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \text{ and } \mathbb{G}$$

$$\tilde{\mathbf{Q}}(\beta) : \begin{cases} \mathbf{b}^{\mathrm{T}}\mathbf{y} \leq \beta \\ \mathbf{P}_i\mathbf{y} \leq \mathbf{q}_i \\ \mathbf{y} \geq \mathbf{0} \end{cases} \tag{LagInner}$$

or (ii) a (compressed) feasible solution for Primal *satisfying* $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq (1 - a_1\epsilon)\beta$ *along with additional properties* $\mathcal{P}rop$*. Then we can implement* ModOracle *using* $O(\rho_i(\log n)\log\frac{1}{\epsilon})$ *queries to* MicroOracle*. Further, we never need to invoke* MicroOracle *for any* $\varrho \geq \frac{12\mathbf{u}_s^{\mathrm{T}}\mathbf{c}}{13\mathbf{z}^{\mathrm{T}}\mathbf{q}_o}$.

Proof. First observe that if MicroOracle answers (ii) then we immediately have our solution for ModOracle. In the remainder we can assume that MicroOracle never answers (ii); observe that is assumed if Dual$(\beta, 1)$ is feasible.

First note that if LagInner is solved, i.e., part (i) applies, then using $O(\log\frac{1}{\epsilon})$ and standard Lagrangian application we can solve Inner. To see that; observe that the solution of MicroOracle also implies a solution of (along with the other constraints) since $\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \geq 0$:

$$\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} - \varrho\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\mathbf{y} \geq \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \frac{13}{12}\varrho\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \tag{9.1.1}$$

We invoke the MicroOracle with $\varrho = 0$. If the returned solution $\mathbf{y}$ satisfies

$$\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\mathbf{y} \leq \frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \tag{9.1.2}$$

then we have immediately a solution for INNER. Now if $\varrho \geq \varrho_0 = \frac{12\mathbf{u}_s^{\mathrm{T}}\mathbf{c}}{13\mathbf{z}^{\mathrm{T}}\mathbf{q}_o}$ then $\mathbf{y} = \mathbf{0}$ is a feasible solution for Equation 9.1.1 since right hand size is 0. Note $\mathbf{y} = \mathbf{0}$ also satisfies $\mathbb{G}$ by assumption and definitely satisfies Equation 9.1.2.

Therefore we can perform a binary search over ϱ and finally achieve an interval $[\varrho_1, \varrho_2]$ where in the corresponding solutions; $\tilde{\mathbf{y}}_1$ does not satisfy Equation 9.1.2 and $\tilde{\mathbf{y}}_2$ does (but both satisfy Equation 9.1.1, $\mathbb{G}$ and $\tilde{\mathbf{Q}}(\beta)$). Let $\Upsilon = \frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o$. Eventually we can narrow the interval $\varrho_2 - \varrho_1 \leq \epsilon\varrho_0$, $\varrho_2 > \varrho_1$ and

$$\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\tilde{\mathbf{y}}_1 = \Upsilon_1 > \Upsilon = \frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o \text{ and } \mathbf{z}^{\mathrm{T}}\mathbf{P}_o\tilde{\mathbf{y}}_2 = \Upsilon_2 \leq \Upsilon$$

We can then find two numbers s_1, s_2 such that $s_1 + s_2 = 1$ and $s_1\Upsilon_1 + s_2\Upsilon_2 = \Upsilon = \frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o$. Let $\mathbf{y} = s_1\tilde{\mathbf{y}}_1 + s_2\tilde{\mathbf{y}}_2$. Observe that $\mathbf{y}$ satisfies Equation 9.1.2, $\mathbb{G}$ and $\tilde{\mathbf{Q}}(\beta)$. Now observe that since $\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2$ both satisfy Equation 9.1.1,

$$\begin{aligned}
\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\mathbf{y} &= s_1\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{y}}_1 + s_2\mathbf{u}_s^{\mathrm{T}}\mathbf{A}\tilde{\mathbf{y}}_2 \\
&\geq s_1\left(\mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho_1\frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o + \varrho_1\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\tilde{\mathbf{y}}_1\right) + s_2\left(\mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho_2\frac{13}{12}\mathbf{z}^{\mathrm{T}}\mathbf{q}_o + \varrho_2\mathbf{z}^{\mathrm{T}}\mathbf{P}_o\tilde{\mathbf{y}}_2\right) \\
&= \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho_1 s_1\left(\Upsilon - \Upsilon_1\right) - \varrho_2 s_2\left(\Upsilon - \Upsilon_2\right) \\
&= \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho_1 s_1\left(\Upsilon - \Upsilon_1\right) - \varrho_1 s_2\left(\Upsilon - \Upsilon_2\right) - \left(\varrho_2 - \varrho_1\right)\left(\Upsilon - \Upsilon_2\right) \\
&= \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \varrho_1\left(\Upsilon - \left(s_1\Upsilon_1 + s_2\Upsilon_2\right)\right) - \left(\varrho_2 - \varrho_1\right)\left(\Upsilon - \Upsilon_2\right) \\
&= \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - 0 - \left(\varrho_2 - \varrho_1\right)\left(\Upsilon - \Upsilon_2\right) \geq \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \epsilon\varrho_0\Upsilon = \mathbf{u}_s^{\mathrm{T}}\mathbf{c} - \epsilon\mathbf{u}_s^{\mathrm{T}}\mathbf{c}
\end{aligned}$$

But observe that INNER is *exactly* the oracle required for solving SPARSE using the fractional packing framework! This requires $O(\rho_i \log n)$ invocations of INNER since the width is ρ_i and we want a $(1 + 6\delta)$ approximate solution of fractional packing

where $\delta = \frac{1}{6}$. Again, if we fail to find any of these $O(\rho_i \log n)$ solutions, then we already have a solution for part (ii) of ModOracle. Note the condition on ϱ follows from the construction. $\square$

In summary we get Theorem 9.1.3.

9.2 The Unweighted b-Matching Problem

In this section we prove the following theorem:

Theorem 9.2.1. *For any $0 < \epsilon \leq 1$ and $p \leq \frac{\log n}{64(\log \frac{1}{\epsilon} + \log\log n)}$ we can construct an integral solution for the unweighted maximum b-Matching to a factor $(1 - O(\epsilon))$ in $O(p/\epsilon)$ passes and $O(pm \operatorname{poly}(\epsilon^{-1}, \log n))$ time and $O(n^{1+1/p})$ space.*

The formulation LP39 is the unweighted version of the standard LP formulation for the b–Matching problem. Recall that variable x_{ij} corresponds to the multiplicity of the edge (i, j) in the optimum solution. We assume that $x_{ij} = 0$ for all $(i, j) \notin E$. Again, this is an undirected formulation so $x_{ij} = x_{ji}$, i.e., the order of the indices of x are not relevant. Also, $||U||_b = \sum_{i \in U} b_i$ and $\mathcal{O} = \{U \subseteq V;\ ||U||_b \text{ is odd}\}$. We have

the following primal and dual programs:

$$\beta^* = \max \sum_{(i,j)} x_{ij}$$

$$\sum_j x_{ij} \leq b_i \qquad \forall i \in V$$

$$\sum_{(i,j):i,j\in U} x_{ij} \leq \lfloor ||U||_b/2 \rfloor \quad \forall U \in \mathcal{O}$$

$$x_{ij} \geq 0 \qquad \text{(LP39)}$$

$$\beta^* = \min \sum_i b_i y_i + \sum_{U\in\mathcal{O}} \lfloor ||U||_b/2 \rfloor z_U$$

$$y_i + y_j + \sum_{U\in\mathcal{O};i,j\in U} z_U \geq 1 \qquad \forall (i,j) \in E$$

$$y_i, z_U \geq 0 \qquad \text{(LP40)}$$

We would like to proceed as in Section 6.3, by altering the formulation LP40 so that it is amenable to the dual-primal method. However that step is significantly more involved. We first use an intermediate formulation.

Lemma 9.2.2. *$\hat{\beta} = \beta^*$ where $\hat{\beta}$ is defined as follows:*

$$\hat{\beta} = \min \sum_i b_i y_i + \sum_{U\in\mathcal{O}} \lfloor ||U||_b/2 \rfloor z_U$$

$$y_i + y_j + \sum_{U\in\mathcal{O};i,j\in U} z_U \geq 1 \quad \forall (i,j) \in E$$

$$2y_i + \sum_{U\in\mathcal{O}:i\in U} z_U \leq 2 \quad \forall i$$

$$y_i, z_U \geq 0 \qquad \text{(LP41)}$$

$$\hat{\beta} = \max \sum_{(i,j)} x_{ij} - 2\sum_i \mu_i$$

$$\sum_j x_{ij} - 2\mu_i \leq b_i \quad \forall i$$

$$\sum_{(i,j):i,j\in U} x_{ij} - \sum_{i\in U} \mu_i \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor \quad \forall U \in \mathcal{O}$$

$$x_{ij}, \mu_i \geq 0 \qquad \text{(LP42)}$$

Proof. Cunningham and Marsh [31] show that the optimum solution $\{y_i^*\}, \{z_U^*\}$ of LP40 is laminar and integral for regular matching. This is proved for b–Matching in [2]. However those result in themselves are not sufficient to prove the lemma.

Start from a laminar integral solution of LP40. Observe that all $y_i^*, z_U^* \leq 1$ since

edge weights are at most 1. Next observe that if $z^*_U = 1$ then we will never have $z^*_{U'} = 1$ for any $U' \subset U$. Therefore $\{U | z^*_U = 1\}$ form a disjoint family.

Finally for an odd set U with $z^*_U = 1$ let $U^* = \{i | y^*_i = 1; i \in U\}$. If $U^* \neq \emptyset$ then we can set $\hat{y}_j = \frac{1}{2}$ for $j \in U \setminus U^*$ and $\hat{y}_i = 1$ for $i \in U^*$. Set $\hat{z}_U = 0$ and for all other sets U' we keep $\hat{z}_{U'} = z^*_{U'}$ and for all vertices $j \notin U$ we have $\hat{y}_j = y^*_j$. Then this new solution $\{\hat{y}_i\}, \{\hat{z}_U\}$ satisfies all the constraints and has a better or equal objective value! Therefore we can assume $\bigcup_{U: z^*_U = 1} U$ and $\{i | y^*_i = 1\}$ are disjoint. Therefore the constraint $2y_i + \sum_{U \in \mathcal{O}: i \in U} z_U \leq 2$ follows for all i. Note that this constraint is true for an *optimum half-integral* solution of LP40. $\square$

The actual formulation used is given by the next lemma:

Lemma 9.2.3. *Let* $\mathcal{O}_s = \{U \subseteq V;\ ||U||_b$ **is odd and** $||U||_b \leq 4/\epsilon\}$. *Consider:*

$$\mathbf{A}: \left\{ y_i + y_j + \sum_{U \in \mathcal{O}_s; i,j \in U} z_U \geq 1 \qquad \forall (i,j) \in E \right.$$

$$\mathbf{Q}(\beta, \Lambda) \left\{ \begin{array}{l} \mathbf{A}'(\Lambda): \left\{ 2y_i + \sum_{U \in \mathcal{O}_s : i \in U} z_U \leq 2\Lambda \qquad \forall i \right. \\ \tilde{\mathbf{Q}}(\beta): \left\{ \begin{array}{l} \sum_i b_i y_i + \sum_{U \in \mathcal{O}_s} \lfloor ||U||_b / 2 \rfloor z_U \leq \beta \\ 2y_i + \sum_{U \in \mathcal{O}_s : i \in U} z_U \leq 4/\epsilon \qquad \forall i \\ y_i, z_U \geq 0 \end{array} \right. \end{array} \right. \tag{LP43}$$

The $(1 - 3\epsilon)$ *feasibility of LP43 for any* $\Lambda \geq 1$, *i.e.,* $y_i + y_j + \sum_{U \in \mathcal{O}_s; i,j \in U} z_U \geq (1 - 3\epsilon), \forall (i,j) \in E$ *implies* $\beta^* \leq \beta/(1 - 3\epsilon)$. *As a consequence the formulation LP42 is* **Dual-Primal amenable** *according to Definition 9.1.1. Note,* $m = n^2$ *still, even though we have* $N = n^{O(1/\epsilon)}$ *variables.*

Proof. If we have a $(1-3\epsilon)$ feasible solution of LP43 we have $\beta^* \leq \beta/(1-3\epsilon)$ simply by scaling and feasibility of LP40 (we have a feasible solution of LP40 only using some of the variables). $\square$

We now prove the following two theorems; which together imply Theorem 9.2.1.

Theorem 9.2.4. *We either provide a feasible (but compressed) solution to LP42 with* $\sum_{(i,j)\in E} x_{ij} - 2\sum_i \mu_i \geq \beta(1-3\epsilon/2)$ *with the additional property* $\sum_i \mu_i \leq 6\beta$ *or we prove that* $\beta \geq (1-6\epsilon)\beta^*$. *For the parameter* p *given in Theorem 9.2.4 we can implement the algorithm using* $O(p/\epsilon)$ *passes,* $O(pm\epsilon^{-4}\log^2 n + n^{1+1/p})$ *time and* $O(n^{1+1/p})$ *space.*

Theorem 9.2.5. *For any* $0 < \epsilon \leq \frac{1}{6}$ *given a feasible (but compressed) solution of LP42 with* $\beta/(1-6\epsilon) \geq \sum_{(i,j)\in E} x_{ij} - 2\sum_i \mu_i \geq \beta(1-3\epsilon/2)$ *and the additional property that* $\sum_i \mu_i \leq 6\beta$ *we can produce an integral b–Matching of size at least* $(1-5\epsilon)\beta$ *using* $O(n \operatorname{poly} \epsilon^{-1} \log n)$ *space and time.*

We prove Theorem 9.2.4 first – this is where the feasible solution is produced. Theorem 9.2.5 is about rounding and is proved in Section 9.2.1. To prove Theorem 9.2.4 we would want to apply Theorem 9.1.3 to Lemma 9.2.3; but first we must show the existence of a deferred sparsification as in Section 6.3; and define the property $\mathbb{G}$ defined in Definition 9.1.4 and used in Theorem 9.1.3.

Lemma 9.2.6. *Suppose we have* $H = (V, E', u^s)$ *as a* $(\epsilon/16)$*-Cut-Sparsifier for* $G = (V, E, u)$. *Define the constraint* $\mathbb{G}$ *to indicate the property that for any* $\tilde{z}_U > 0$ *the set*

U satisfies $\sum_{(i,j):i,j\in U} u^s_{ij} \geq \sum_{i\in U}\sum_{j\notin U} u^s_{ij} = \mathcal{C}(U,u^s)$. Then,

$$\sum_i \tilde{y}_i \sum_j u^s_{ij} + \sum_{U\in\mathcal{O}_s} \tilde{z}_U \sum_{(i,j):i,j\in U} u^s_{ij} \geq (1-\frac{\epsilon}{8})\sum_{(i,j)} u^s_{ij} \Longrightarrow$$

$$\sum_i \tilde{y}_i \sum_j u_{ij} + \sum_{U\in\mathcal{O}_s} \tilde{z}_U \sum_{(i,j):i,j\in U} u_{ij} \geq (1-\frac{\epsilon}{2})\sum_{(i,j)} u_{ij}$$

Proof. Observe that $(1-\epsilon/16)\sum_j u^s_{ij} \leq \sum_j u_{ij} \leq (1+\epsilon/16)\sum_j u^s_{ij}$ using a cut defined on the single vertex. Note

$$\begin{aligned}\sum_{(i,j)} u^s_{ij} &= \frac{1}{2}\sum_i\sum_j u^s_{ij} \geq \frac{1}{2}\sum_i\left(\sum_j u_{ij}\right)\frac{1}{(1+\epsilon/8)}\\ &\geq \left(1-\frac{\epsilon}{8}\right)\frac{1}{2}\sum_i\sum_j u_{ij} = \left(1-\frac{\epsilon}{8}\right)\sum_{(i,j)} u_{ij}\end{aligned}$$

Also for any U with $\tilde{z}_U > 0$ we have:

$$\begin{aligned}\sum_{(i,j):i,j\in U} u_{ij} &= \frac{1}{2}\left(\sum_{i\in U}\sum_j u_{ij} - \mathcal{C}(U,u)\right)\\ &\geq \frac{1}{2}\left(\sum_{i\in U}\left(1-\frac{\epsilon}{16}\right)\sum_j u^s_{ij} - \left(1+\frac{\epsilon}{16}\right)\mathcal{C}(U,u^s)\right)\\ &= \left(1-\frac{\epsilon}{16}\right)\frac{1}{2}\left(\sum_{i\in U}\sum_j u^s_{ij} - \mathcal{C}(U,u^s)\right) - \frac{\epsilon}{16}\mathcal{C}(U,u^s)\\ &\geq \left(1-\frac{\epsilon}{16}\right)\sum_{(i,j):i,j\in U} u^s_{ij} - \frac{\epsilon}{16}\sum_{(i,j):i,j\in U} u^s_{ij}\\ &\qquad\qquad \left(\text{since } \textstyle\sum_{(i,j):i,j\in U} u^s_{ij} \geq \mathcal{C}(U,u^s)\right)\end{aligned}$$

which implies $\sum_{(i,j):i,j\in U} u_{ij} \geq \left(1-\frac{\epsilon}{8}\right)\sum_{(i,j):i,j\in U} u^s_{ij}$. Therefore

$$\begin{aligned}
\sum_i \tilde{y}_i \sum_j u_{ij} + \sum_{U\in\mathcal{O}_s} \tilde{z}_U \sum_{(i,j):i,j\in U} u_{ij} \\
&\geq \sum_i \tilde{y}_i \left(1-\frac{\epsilon}{16}\right)\sum_j u^s_{ij} + \sum_{U\in\mathcal{O}_s} \tilde{z}_U \left(1-\frac{\epsilon}{8}\right) \sum_{(i,j):i,j\in U} u^s_{ij} \\
&\geq \left(1-\frac{\epsilon}{8}\right)\left(\sum_i \tilde{y}_i \sum_j u^s_{ij} + \sum_{U\in\mathcal{O}_s} \tilde{z}_U \sum_{(i,j):i,j\in U} u^s_{ij}\right) \\
&\geq \left(1-\frac{\epsilon}{4}\right)\left(1-\frac{\epsilon}{8}\right)\sum_{(i,j)} u^s_{ij} \\
&\geq \left(1-\frac{\epsilon}{4}\right)\left(1-\frac{\epsilon}{8}\right)\left(1-\frac{\epsilon}{8}\right)\sum_{(i,j)} u_{ij} \geq \left(1-\frac{\epsilon}{2}\right)\sum_{(i,j)} u_{ij}
\end{aligned}$$

The lemma follows. □

The algorithm for constructing a deferred sparsification will be given in Section 9.4. Theorem 9.2.4 follows from Lemma 9.2.8 and Theorem 9.1.3. We need Lemma 7.5.1.

Lemma 7.5.1. *Given an unweighted graph G with parameters κ and a special node s, in time $O(n\,\mathrm{poly}(\delta^{-1}, \log n))$ we can identify a collection $\mathcal{L}$ of odd-sets which (i) do not contain s (ii) define cut of at most κ in G and (iii) every other odd set not containing s and with a cut less than κ intersects with a set in $\mathcal{L}$.*

We will be using a corollary of the above:

Lemma 9.2.7. *Given a graph $G = (V, E)$, non-negative numbers q_{ij} on edges and $\hat{q}_i$ on nodes $i \in V$ such that*

(A1) If $b_i = 1$ then $\hat{q}_i \geq C$.

(A2) $\sum_j q_{ij} \leq \hat{q}_i$ for all i.

(A3) Any odd-set U with $||U||_b > 4/\epsilon$ satisfies $\sum_{i\in U}\left(\hat{q}_i - \sum_j q_{ij}\right) > 1$.

using space $O(n\epsilon^{-2})$ and $O(m)+O(n\,\mathrm{poly}(\epsilon^{-1},\log n))$ time and a single pass over the list of edges we can find a collection $\mathcal{L}$ such that (i) every odd-set $U \in \mathcal{L}$ satisfies

$$\sum_{(i,j):i,j\in U} q_{ij} \geq \frac{1}{2}\left(\sum_{i\in U}\hat{q}_i - 1\right) \tag{9.2.1}$$

and (ii) every odd set $U \notin \mathcal{L}$ either intersects with a set in $\mathcal{L}$ or satisfies

$$\sum_{(i,j):i,j\in U} q_{ij} \leq \frac{1}{2}\left(\sum_{i\in U}\hat{q}_i - (1-\epsilon)\right) \tag{9.2.2}$$

Observe the conditions (A1) and (A3) imply that singleton vertices or very large sets cannot be present in $\mathcal{L}$ and be returned.

Proof. Follows from creating a graph H on the vertex set $V \cup \{s\}$ where we have $\lfloor q_{ij}8\epsilon^{-3}\rfloor$ parallel edges between i and j. After the edges have been added we add edges between i and s till the degree of s is $\lceil \hat{q}_i 8\epsilon^{-3}\rceil$ — this is feasible due to (A2).

We now apply Lemma 7.5.1 with $\kappa = \lfloor 8\epsilon^{-3}\rfloor$. Note that any set which is returned in $\mathcal{L}$ satisfies Equation 9.2.1 easily since

$$\sum_{(i,j):i,j\in U}\lfloor q_{ij}8\epsilon^{-3}\rfloor \geq \frac{1}{2}\left(\sum_{i\in U}\lceil \hat{q}_i 8\epsilon^{-3}\rceil - \lfloor 8\epsilon^{-3}\rfloor\right) \Longrightarrow \sum_{(i,j):i,j\in U} q_{ij} \geq \frac{1}{2}\left(\sum_{i\in U}\hat{q}_i - 1\right)$$

For any odd set which is not returned and does not intersect the any of the sets returned, the cut after discretization is at least κ;

$$\sum_{(i,j):i,j\in U}\lfloor q_{ij}8\epsilon^{-3}\rfloor \leq \frac{1}{2}\left(\sum_{i\in U}\lceil \hat{q}_i 8\epsilon^{-3}\rceil - \lfloor 8\epsilon^{-3}\rfloor\right)$$
$$\Longrightarrow \sum_{(i,j):i,j\in U} q_{ij} \leq \frac{\binom{4/\epsilon}{2}}{8\epsilon^{-3}} + \frac{1}{2}\left(\sum_{i\in U}\hat{q}_i + \frac{4/\epsilon}{8\epsilon^{-3}} - 1\right)$$

which gives us Equation 9.2.2. □

We now prove the property required by the MicroOracle in Theorem 9.1.3.

Lemma 9.2.8. *Given any nonnegative* $\{u^s_{ij}\}, \{\zeta_i\}, \beta, \epsilon$, *for any* $0 \le \varrho < \frac{6\sum_{(i,j)} u^s_{ij}}{13\sum_i \zeta_i}$, *we can either (i) provide* $\{y_i\}, \{z_U\}$ *that satisfies LP44*

$$\sum_i x_i \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right) + \sum_{U\in\mathcal{O}_s} z_U \left(\sum_{(i,j):i,j\in U} u^s_{ij} - \sum_{i\in U} \varrho\zeta_i\right) + 2\varrho\sum_i \zeta_i \ge \sum_{(i,j)} u^s_{ij}$$

$$\text{subject to } \tilde{\mathbf{Q}}(\beta) \text{ and } \mathbb{G} \tag{LP44}$$

or (ii) provide a compressed representation of a solution for LP42 such that $\sum_{(i,j)} x_{ij} - 2\sum_i \mu_i \ge (1-3\epsilon/2)\beta$ *with the property* $\sum_i \mu_i \le 6\beta$ *(which corresponds to* $\mathcal{P}rop$ *in Theorem 9.1.3).*

Proof. Set $\gamma' = \left(\sum_{(i,j)} u^s_{ij} - 2\varrho\sum_i \zeta_i\right)$. Note that

$$\gamma' \ge \left(\sum_{(i,j)} u^s_{ij} - \frac{12}{13}\sum_{(i,j)} u^s_{ij}\right) = \frac{1}{13}\sum_{(i,j)} u^s_{ij} \ge \frac{1}{6}\sum_i \varrho\zeta_i \tag{9.2.3}$$

First we compute $\mathcal{S} = \{i | \sum_j u^s_{ij} - 2\varrho\zeta_i > \gamma' b_i/\beta\}$ and $\Delta_{\mathcal{S}} = \sum_{i\in\mathcal{S}} \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right)$ If $\Delta_{\mathcal{S}} \ge \epsilon\gamma'/4$ then we set $y'_i = \frac{\gamma'}{\Delta_{\mathcal{S}}}$ for all $i \in \mathcal{S}$ and set all other y'_i, z'_U to 0. It is easy to see that

$$\left(\sum_{(i,j)} u^s_{ij} - 2\varrho\sum_i \zeta_i\right) = \gamma' = \sum_{i\in\mathcal{S}} y'_i \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right)$$

$$\gamma' = \sum_{i\in\mathcal{S}} y'_i \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right) > \sum_{i\in\mathcal{S}} y'_i \frac{\gamma'}{\beta} b_i \quad \Longrightarrow \quad \sum_{i\in\mathcal{S}} y'_i b_i < \beta$$

and $\{y'_i\}, \{z'_U\}$ satisfy the system LP44. Note that no odd sets are used. Therefore in the remainder we assume that $\Delta_{\mathcal{S}} < \epsilon\gamma'/4$. To apply Lemma 9.2.7 on the vertex

set $V \setminus \mathcal{S}$ set:

$$q_{ij} = (1-\epsilon/4)\beta u^s_{ij}/\gamma' \text{ if both } i,j \in V \setminus \mathcal{S} \text{ and 0 otherwise}$$

$$\hat{q}_i = b_i + 2(1-\epsilon/4)\varrho\beta\zeta_i/\gamma'$$

Observe that:

$$\left(\sum_{(i,j):i,j\in U} \frac{(1-\epsilon/4)\beta u^s_{ij}}{\gamma'}\right) \geq \frac{1}{2}\left(\sum_{i\in U}\left(b_i + \frac{2(1-\epsilon/4)\varrho\beta\zeta_i}{\gamma'}\right) - X\right)$$
$$\iff \frac{(1-\epsilon/4)\beta}{\gamma'}\left(\sum_{(i,j):i,j\in U} u^s_{ij} - \sum_{i\in U}\varrho\zeta_i\right) \geq \frac{1}{2}\left(\sum_{i\in U} b_i - X\right) \tag{9.2.4}$$

Observe that we satisfy the conditions **A1**, **A2** trivially. We also satisfy **A3** since for any $i \in V \setminus \mathcal{S}$ we have $\sum_j u^s_{ij} - 2\varrho\zeta_i \leq \gamma' b_i/\beta$, and for $||U||_b > 4/\epsilon$ we have:

$$\begin{aligned}
\sum_{i\in U}\left(\hat{q}_i - \sum_j q_{ij}\right) &= \sum_{i\in U}\left(b_i + \frac{2(1-\epsilon/4)\varrho\beta\zeta_i}{\gamma'} - \sum_j \frac{(1-\epsilon/4)\beta u^s_{ij}}{\gamma'}\right) \\
&= \sum_{i\in U}\left(b_i - \left(1-\frac{\epsilon}{4}\right)\frac{\beta}{\gamma'}\left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right)\right) \\
&\geq \sum_{i\in U} \epsilon b_i/4 = \epsilon||U||_b/4 > 1
\end{aligned}$$

Using Lemma 9.2.7 we get a collection of odd-sets $\mathcal{K}$ where each $U \in \mathcal{K}$ is a subset of $V \setminus \mathcal{S}$ and satisfies the condition $\sum_{(i,j):i,j\in U} q_{ij} \geq \frac{1}{2}\left(\sum_{i\in U}\hat{q}_i - 1\right)$. Using Equation 9.2.4 for $X = 1$ and the fact that $\sum_{i\in U} b_i - 1 = \lfloor ||U||_b/2 \rfloor \geq \frac{1}{3}\sum_{i\in U} b_i$:

$$\begin{aligned}
\left(\sum_{(i,j):i,j\in U} u^s_{ij} - \sum_{i\in U}\varrho\zeta_i\right) &> \frac{\gamma'}{\beta}\left\lfloor \frac{||U||_b}{2}\right\rfloor \geq \frac{1}{3}\sum_{i\in U}\frac{\gamma' b_i}{\beta} \\
&\geq \frac{1}{3}\sum_{i\in U}\left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right)
\end{aligned} \tag{9.2.5}$$

since we have $\sum_j u^s_{ij} - 2\varrho\zeta_i \leq \gamma' b_i/\beta$ for $i \in V \setminus \mathcal{S}$.

Let $\Delta_{\mathcal{K}} = \sum_{U \in \mathcal{K}} \left(\sum_{(i,j):i,j \in U} u^s_{ij} - \varrho \sum_{i \in U} \zeta_i \right)$. If $\Delta_{\mathcal{K}} \geq \epsilon\gamma/8$ then we set $z'_U = \gamma'/\Delta_{\mathcal{K}}$ and otherwise, we set $y'_i, z'_U = 0$. Observe that

$$\sum_i y'_i \left(\sum_j u^s_{ij} - 2\varrho\zeta_i \right) + \sum_{U \in \mathcal{O}_s} z'_U \left(\sum_{(i,j):i,j \in U} u^s_{ij} - 2\varrho \sum_{i \in U} \zeta_i \right)$$
$$= \sum_{U \in \mathcal{K}} \frac{\gamma'}{\Delta_{\mathcal{K}}} \left(\sum_{(i,j):i,j \in U} u^s_{ij} - \varrho \sum_{i \in U} \zeta_i \right) = \frac{\gamma'}{\Delta_{\mathcal{K}}} \Delta_{\mathcal{K}} = \gamma'$$

$$\sum_i y'_i b_i + \sum_{U \in \mathcal{O}_s} z'_U \lfloor ||U||_b/2 \rfloor = \sum_{U \in \mathcal{K}} \frac{\gamma'}{\Delta_{\mathcal{K}}} \lfloor ||U||_b/2 \rfloor$$
$$\leq \sum_{U \in \mathcal{K}} \frac{\gamma'}{\Delta_{\mathcal{K}}} \frac{\beta}{\gamma'} \left(\sum_{(i,j):i,j \in U} u^s_{ij} - \varrho \sum_{i \in U} \zeta_i \right) = \beta$$

$$\left(\sum_{(i,j):i,j \in U} u^s_{ij} - \sum_{i \in U} \varrho\zeta_i \right) \geq \frac{1}{3} \left(\sum_{i \in U} u^s_{ij} - \sum_{i \in U} 2\varrho\zeta_i \right)$$
$$\implies \sum_{(i,j):i,j \in U} u^s_{ij} \geq \sum_{i \in U} \sum_{j \notin U} u^s_{ij} + \sum_{i \in U} \varrho\zeta_i \geq \sum_{i \in U} \sum_{j \notin U} u^s_{ij}$$

We use Equation 9.2.5 in the last two equations. Therefore $\{y'_i\}, \{z'_U\}$ satisfy LP44.

Thus the only case we are left to discuss is $\Delta_{\mathcal{K}} < \epsilon\gamma'/8$ and $\Delta_{\mathcal{S}} \leq \epsilon\gamma'/4$. First, "delete" the edges/vertices associated with $\mathcal{S}$ and $\mathcal{K}$ that is $i \in \mathcal{S} \cup V(\mathcal{K})$ where $V(\mathcal{K}) = \bigcup_{U \in \mathcal{K}} U$:

$$\boldsymbol{\hat{u}_{ij} = u^s_{ij}} \text{ if } i, j \notin \mathcal{S} \cup V(\mathcal{K}) \text{ and } \mathbf{0} \text{ otherwise}$$

$$\boldsymbol{\hat{\zeta}_i = \zeta_i} \text{ if } i \notin \mathcal{S} \cup V(\mathcal{K}) \text{ and } \mathbf{0} \text{ otherwise}$$

Observe that $\mathcal{S}$ and $V(\mathcal{K})$ are disjoint; and

$$\begin{aligned}
\sum_{(i,j)} \hat{u}_{ij} - 2\varrho \sum_i \hat{\zeta}_i &= \sum_{(i,j):i,j\notin\mathcal{S}\cup V(\mathcal{K})} u^s_{ij} - 2\varrho \sum_{i\notin\mathcal{S}\cup V(\mathcal{K})} \zeta_i \\
&= \left(\sum_{(i,j)} u^s_{ij} - 2\varrho \sum_i \zeta_i\right) - \sum_{i\in\mathcal{S}\cup V(\mathcal{K})} \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right) \\
&= \gamma' - \sum_{i\in\mathcal{S}} \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right) - \sum_{U\in V(\mathcal{K})} \sum_{i\in U} \left(\sum_j u^s_{ij} - 2\varrho\zeta_i\right) \\
&\geq \gamma' - \Delta_{\mathcal{S}} - 3\Delta_{\mathcal{K}} \\
&\qquad \text{(Using Equation 9.2.5 and definition of } \Delta_{\mathcal{K}} \text{)} \\
&\geq \gamma'(1 - \epsilon/4 - 3\epsilon/8) = \gamma'(1 - 5\epsilon/8) \qquad (9.2.6)
\end{aligned}$$

Now set $x_{ij} = \frac{(1-\epsilon/4)\hat{u}_{ij}\beta}{(1+\epsilon/2)\gamma'}$ and $\mu_i = \frac{(1-\epsilon/4)\varrho\hat{\zeta}_i\beta}{(1+\epsilon/2)\gamma'}$, which implies that:

$$\begin{aligned}
\sum_{(i,j)} x_{ij} - 2\sum_i \mu_i &\geq \frac{(1-\epsilon/4)\beta}{(1+\epsilon/2)\gamma'} \left(\sum_{(i,j)} \hat{u}_{ij} - 2\varrho \sum_i \hat{\zeta}_i\right) \\
&\geq \frac{(1-\epsilon/4)\beta}{(1+\epsilon/2)\gamma'} \gamma'(1 - 5\epsilon/8) > \beta(1 - 3\epsilon/2)
\end{aligned}$$

Moreover (Using Equation 9.2.3) we have:

$$\sum_i \mu_i \leq \sum_i \frac{(1-\epsilon/4)\varrho\hat{\zeta}_i\beta}{(1+\epsilon/2)\gamma'} \leq \frac{(1-\epsilon/4)\beta}{(1+\epsilon/2)} \frac{\sum_i \varrho\zeta_i}{\gamma'} \leq \frac{(1-\epsilon/4)\beta}{(1+\epsilon/2)} 6 \leq 6\beta$$

To verify that $\{x_{ij}\}, \{\mu_i\}$ satisfy LP42, observe that for any $i \in \mathcal{S} \cup V(\mathcal{K})$ the constraints are satisfied trivially since $y_{ij} = \mu_i = 0$. Moreover if an $i \in \mathcal{S} \cup V(\mathcal{K})$ causes a violation of any odd-set U then either $||U - \{i\}||_b$ is odd (which implies there is a smaller violated odd-set) or

$$\sum_{(i',j'):i',j\in U-\{i\}} x_{i'j} - \sum_{i'\in U-\{i\}} \mu_{i'} > \frac{||U - \{i\}||_b}{2} \Longrightarrow \sum_{i'\in U-\{i\}} \left(\sum_j x_{i'j} - 2\mu_{i'} - b_{i'}\right) > 0$$

which implies that there is a vertex violation. Therefore it suffices to focus on $V \setminus (\mathcal{S} \cup V(\mathcal{K}))$.

Observe that for any $i \notin \mathcal{S}$ (and therefore for all $i \in V \setminus (\mathcal{S} \cup V(\mathcal{K}))$) we have:

$$\sum_j x_{ij} - 2\mu_i \leq \frac{(1-\epsilon/4)\beta}{(1+\epsilon/2)\gamma'} \left(\sum_j u^s_{ij} - 2\varrho\zeta_i \right) \leq \frac{(1-\epsilon/4)}{(1+\epsilon/2)} b_i$$

Therefore all vertex constraints are satisfied. For $U \subseteq V \setminus (\mathcal{S} \cup V(\mathcal{K}))$, observe that if $||U||_b \geq 4/\epsilon$ then:

$$\sum_{i \in U} \left(\sum_j x_{ij} - 2\mu_i \right) \leq \frac{(1-\epsilon/4)}{(1+\epsilon/2)} ||U||_b \leq ||U||_b - 1 \Longrightarrow \sum_{(i,j):i,j \in U} x_{ij} - \sum_{i \in U} \mu_i \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor$$

Therefore all that remain to be verified are odd-sets $U \subseteq V \setminus (\mathcal{S} \cup V(\mathcal{K}))$ with $||U||_b < 4/\epsilon$. Using Equation 9.2.2 from Lemma 9.2.7 (and Equation 9.2.4 with $X = 1 - \epsilon$) for any small odd-set $U \subseteq V \setminus \mathcal{S}$ disjoint from $\mathcal{K}$ we have:

$$\begin{aligned} \sum_{(i,j):i,j \in U} x_{ij} - 2\sum_{i \in U} \mu_i &\leq \frac{1}{(1+\epsilon/2)} \frac{1}{2} \left(\sum_i b_i - (1-\epsilon) \right) = \frac{1}{(1+\epsilon/2)} \frac{1}{2} \left(\sum_i b_i - 1 + \epsilon \right) \\ &= \frac{1}{(1+\epsilon/2)} \left(\left\lfloor \frac{||U||_b}{2} \right\rfloor + \epsilon/2 \right) \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor \end{aligned}$$

The lemma follows. □

9.2.1 Proof Of Theorem 9.2.5

Theorem 9.2.5. *For any $0 < \epsilon \leq \frac{1}{6}$ given a feasible solution of LP42 with $\beta/(1-6\epsilon) \geq \sum_{(i,j)\in E} x_{ij} - 2\sum_i \mu_i \geq \beta(1-3\epsilon/2)$ and the additional property that $\sum_i \mu_i \leq 6\beta$ we can produce an integral b–Matching of size at least $(1-5\epsilon)\beta$ using $O(n$*

$\text{poly}(\epsilon^{-1}, \log n))$ *space and time.*

$$\begin{aligned}
&\hat{\beta} = \max \sum_{(i,j)} x_{ij} - 2\sum_i \mu_i \\
&\sum_j x_{ij} \leq b_i + 2\mu_i && \forall i \in V \\
&\sum_{(i,j):i,j\in U} x_{ij} \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor + \sum_{i\in U} \mu_i && \forall U \in \mathcal{O} \\
&x_{ij}, \mu_i \geq 0 && \text{(LP42)}
\end{aligned}$$

Proof. Set $x'_{ij} = (1-\epsilon)x_{ij}$ and $\mu'_i = (1-\epsilon)\mu_i$ if $\mu_i \geq \epsilon/4$ and $\mu'_i = 0$ otherwise. Observe that $\{x'_{ij}\}, \{\mu'_i\}$ represent a feasible solution of LP42. Let $V' = \{i|\mu'_i > 0\}$. Note since $\sum_i \mu_i \leq 6\beta$ under the assumption in the theorem, we can immediately conclude that $|V'| \leq 24\beta/\epsilon$. Note $\sum_{(i,j)\in E} x'_{ij} - \sum_i \mu'_i \geq (1-3\epsilon)\beta$.

For each $i \in V'$ add a new node designated as the "mate" of i with $b = 2$. Add an edge between i and its mate and choose that new edge with fractional weight $2(\lceil \mu'_i \rceil - \mu'_i)$. Let the set of mates V'' and this new solution be $\{\hat{x}_{ij}\}$. Note:

$$\begin{aligned}
\sum_{(i,j)} \hat{x}_{ij} &= \sum_{(i,j)} x'_{ij} + \sum_i 2(\lceil \mu'_i \rceil - \mu'_i) \leq \sum_{(i,j)} x'_{ij} + 2|V'| \\
&\leq (\sum_{(i,j)} x'_{ij} - 2\sum_i \mu'_i) + 2\sum_i \mu'_i + \frac{48\beta}{\epsilon} < \frac{50\beta}{\epsilon}
\end{aligned}$$

Since we are adding nodes with even b values; these new nodes cannot create any violated odd-sets in the new graph (otherwise we can remove the new nodes and continue to preserve the violations). Therefore $\{\hat{x}_{ij}\}$ is a feasible b–Matching in this new graph with capacities $b_i + \lceil \mu'_i \rceil$ for the old vertices in V and 2 for vertices in V''.

Now perform the rounding of $\{\hat{x}_{ij}\}$ to a factor $(1-\epsilon^2/25)$ as discussed in Theorem 9.5.1 in Section 9.5 (setting $\varepsilon = \epsilon^2/125$). Discard $2\lceil\mu'_i\rceil$ edges from each $i \in V'$ as follows. First discard any edge from a node in V to its mate in V''. At this point we have no edges incident to any vertex in V'' and have deleted at most 2 edges per vertex. We now delete the remainder of the edges arbitrarily, such that $2\lceil\mu'_i\rceil$ edges in total are deleted from each $i \in V'$. We have an integral b–Matching which respects the capacities b_i and whose value is at least $(1-\epsilon^2/25)\sum_{(i,j)}\hat{x}_{ij} - \sum_i 2\lceil\mu'_i\rceil$. Observe that

$$\begin{aligned}(1-\epsilon^2/25)\sum_{(i,j)}\hat{y}_{ij} - \sum_i 2\lceil\mu'_i\rceil &\geq \sum_{(i,j)}\hat{y}_{ij} - \sum_i 2\lceil\mu'_i\rceil - 2\epsilon\beta \\ &= \sum_{(i,j)} y'_{ij} - \sum_i 2\mu'_i - 2\epsilon\beta \geq (1-3\epsilon)\beta - 2\epsilon\beta\end{aligned}$$

Theorem 9.2.5 follows. □

9.3 The Weighted b-Matching Problem

In this section we prove the following theorem; we remark that the main point of the theorem is that the number of passes is independent of n and the running time is near linear in m.

Theorem 9.3.1. *For any $0 < \epsilon \leq 1$ and $p \leq \frac{\log n}{64(\log\frac{1}{\epsilon}+\log\log n)}$ we can construct a compressed fractional solution for the maximum unweighted b-Matching to a factor $(1-9\epsilon)$ in $O(p\epsilon^{-13}\log^2\frac{1}{\epsilon}) = O(p\,\mathrm{poly}\,\epsilon^{-1})$ passes (or rounds) and $O(m\,\mathrm{poly}(\epsilon^{-1}, \log B))$ time and $O(n^{1+1/p})$ space.*

New ideas needed and Roadmap: The algorithm and the proof of Theorem 9.3.1 will follow the discussion in Sections 6.3 and Section 9.2 but with major differences.

(i) We implement an initial "pruning" step which removes very low weight edges which are unlikely to impact any maximal matching. This step is critical to ensure that the covering program has low width. This is discussed in Section 9.3.1.

(ii) The covering program we formulate is significantly different from previous sections. The key intuition is to restrict the space of odd-sets further (we already saw $||U||_b \leq 4/\epsilon$). This is also critical in ensuring that the width of any formulation is small. This restriction along with the resulting fractional covering formulation is discussed in Section 9.3.2.

(iii) The Oracle is much more complicated than in Sections 6.3 and 9.2. Intuitively the oracle in those sections used the fact that either there were offending vertices or there were offending odd-sets; and removal of both gave an almost feasible solution. However we now have different tiers of weight, and the number of tiers depend on n. If we want to ensure that the width is independent of n then we have to analyze multiple tiers of weight simultaneously. This implies that Lemma 9.2.7 is invoked multiple times; which in turn indicates why isolation was necessary (to ensure that the different invocations of Lemma 9.2.7 refer to the same quantities). This is described in Section 9.3.4.

Step 1: Bounding the smallest edge weight and Discretizing Weights. We throw away all edges smaller than $\epsilon W/B$ where $B = \sum_i b_i$ and W is the largest weight. We then scale the edge weight such that the smallest edge weight W_m is scaled to 1. We then round the weight of every edge **down** to the nearest power of $(1+\epsilon)$. Let L denote the number of different edge weights we have: $L = O(\frac{1}{\epsilon}\log B)$. It is immediate that the maximum weighted b–Matching in this new graph G', denoted by $\mathcal{M}(G')$, satisfies $\mathcal{M}(G) \geq W_m\mathcal{M}(G') \geq \frac{(1-\epsilon)}{(1+\epsilon)}\mathcal{M}(G) > (1-2\epsilon)\mathcal{M}(G)$. Therefore estimating $\mathcal{M}(G')$ to $(1-O(\epsilon))$ factor gives us a $(1-O(\epsilon))$ approximate estimate for $\mathcal{M}(G)$.

Lemma 9.3.2. *Given any input graph $G' = (V, E')$ and a b-Matching instance which has been discretized as in Step 1, we can compute a graph $\tilde{G}$ and a corresponding b-Matching instance and vertex weights $\{\varpi_i\}$ such that $\mathcal{M}(G') \geq \mathcal{M}(\tilde{G}) \geq (1-4\epsilon)\mathcal{M}(G')$ and the edge weight of an edge w_{ij} in $\tilde{G}$ incident to the vertex i, is in the range $[\frac{\varpi_i}{\vartheta}, \varpi_i]$ where $\vartheta = \frac{8}{\epsilon^2}\left\lceil\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\right\rceil$. We use $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$ passes for this and each w_{ij} remains at least 1. The number of vertices $\tilde{V}$ in $\tilde{G}$ is at most $O(\frac{n}{\epsilon}\log B)$ and the number of edges $\tilde{E}$ is at most $O(\frac{m}{\epsilon^2}\log^2\frac{1}{\epsilon})$ and $\sum_{i\in\tilde{V}}\varpi_i \leq 6\epsilon\vartheta\mathcal{M}(\tilde{G})$.*

Observe that $\tilde{G}$ will not have integer weights, but it will not be relevant due to the following non-trivial lemma – note Cunningham-Marsh [31] assumes integer weights.

Lemma 9.3.3. *Let $\mathfrak{O} = \{U \subseteq \tilde{V};\ ||U||_b$ is odd, $||U||_b \leq \frac{4}{\epsilon}, 3\vartheta\varpi_j \geq \epsilon^3\varpi_i$ for all $i, j \in$*

$U\}$.

$$\begin{aligned}
\hat{\beta} = \min \quad & \textstyle\sum_i b_i y_i + \sum_{U \in \mathfrak{O}} \lfloor ||U||_b/2 \rfloor z_U \\
s.t \quad & y_i + y_j + \textstyle\sum_{U \in \mathfrak{O}; i,j \in U} z_U \geq w_{ij} && \forall (i,j) \in \tilde{E} \\
& 2y_i + \textstyle\sum_{U \in \mathfrak{O}; i \in U} z_U \leq 2\varpi_i && \forall i \in \tilde{V} \\
& y_i, z_U \geq 0 && \forall i \in \tilde{V}; \forall U \in \mathfrak{O}
\end{aligned} \tag{LP45}$$

There exists a feasible solution $\{\hat{y}_i\}, \{\hat{z}_U\}$ for LP45 such that the objective value is $\hat{\beta} \leq (1+\epsilon)\mathcal{M}(\tilde{G})$ and $L = \{U | \hat{z}_U \neq 0\}$ is a laminar family.

The above lemma is completed by the complementary lemma regarding its dual;

Lemma 9.3.4. *Given a (compressed; i.e., specified by a streaming algorithm) solution $\{x_{ij}\}$ to the system LP46,*

$$\begin{aligned}
& \textstyle\sum_{(i,j)} w_{ij} x_{ij} = \beta \\
& \textstyle\sum_j x_{ij} \leq b_i && \forall i \\
& \sum_{(i,j): i,j \in U} x_{ij} \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor && \forall U \in \mathfrak{O} \\
& x_{ij} \geq 0
\end{aligned} \tag{LP46}$$

we can produce a solution (a transducer which produces the solution on reading the streaming input one more time) $\{\hat{x}_{ij}\}$ using $O(\epsilon^{-3})$ passes and $O(n \operatorname{poly}(\epsilon^{-1}, \log n))$ space such that $\sum_{(i,j)} w_{ij}\hat{x}_{ij} = (1 - O(\epsilon))\beta$ and the constraints $\sum_j x_{ij} \leq b_i$ for all i; and $\sum_{(i,j): i,j \in U} x_{ij} \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor$ for all $U \in \mathcal{O} = \{U | ||U||_b$ is odd $\}$ (not just $\mathfrak{O}$) hold, i.e., the system LP22 is satisfied. Note that this implies a feasible fractional solution for the b–Matching problem on $\tilde{G}$.

On the basis of Lemmas 9.3.3 and 9.3.4, we formulate the following fractional covering problem where $\rho = 24\vartheta^2/\epsilon^4 = O(\epsilon^{-12}\log^2\frac{1}{\epsilon})$:

$$\mathbf{A}: \left\{ y_i + y_j + \sum_{U\in\mathfrak{O};i,j\in U} z_U \geq w_{ij} \qquad \forall (i,j)\in\tilde{E} \right.$$

$$\mathcal{P}_w : \begin{cases} \sum_i b_i y_i + \sum_{U\in\mathfrak{O}} \lfloor ||U||_b/2 \rfloor z_U \leq \beta \\ y_i + y_j + \sum_{U\in\mathfrak{O};i,j\in U} z_U \leq \rho w_{ij} \quad \forall (i,j)\in\tilde{E} \\ y_i, z_U \geq 0 \end{cases} \tag{LP47}$$

The final two lemmas complete Theorem 9.3.1.

Lemma 9.3.5. *Given a deferred sparsification $\{u^s_{ij}\}$ of $\{u_{ij}\}$, a solution to LP48 such that in addition we satisfy that or any $U\in\mathfrak{O}$ such that $z'_U > 0$ we have $\sum_{i,j\in U} u^s_{ij} \geq \sum_{i\in U}\sum_{j\notin U} u^s_{ij} = \mathcal{C}(U, u^s)$,*

$$\sum_i y'_i \sum_j u^s_{ij} + \sum_{U\in\mathfrak{O}} z'_U \sum_j u^s_{ij} \geq \sum_{(i,j)\in\tilde{E}} u^s_{ij} w_{ij} \text{ subject to } \mathcal{P}_w \tag{LP48}$$

also satisfy LP49 with $y_i = y'_i$ and $z_U = z'_U$;

$$\sum_i y_i \sum_j u_{ij} + \sum_{U\in\mathfrak{O}} z_U \sum_{i,j\in U} u_{ij} \geq \left(1 - \frac{\epsilon}{2}\right) \sum_{(i,j)\in\tilde{E}} u_{ij} w_{ij} \text{ subject to } \mathcal{P}_w \tag{LP49}$$

Note that solutions of LP49 are collected to produce a feasible solution to LP47. This algorithm runs in time $O(n^{1+1/(2p)})$ and space.

Lemma 9.3.6. *Given an instance of LP48 we either (i) produce a solution or a (ii) feasible solution for LP46 in time and space $O(n\,\mathrm{poly}(\log B, \epsilon^{-1}))$.*

Observe that we are **not** using a two step dual-primal algorithm, rather a one step approach – primarily due to the simplicity of the one step process. It seems a two-step

approach is feasible, exactly along the lines of Section 9.2, however the rounding will be a bit more involved. Moreover in that case the number of passes will reduce to $O(p\vartheta/\epsilon^4)$ — which is still best thought of as $O(\text{poly}\,\epsilon^{-1})$ and the overall characteristic of the algorithm does not change significantly to merit the more detailed discussion.

However the main point of Theorem 9.3.1 is that we can solve maximum weighted b–Matching in a streaming setting using passes (or rounds in a map-reduce setting) which are polynomial in $1/\epsilon$. The best results to date, for the weighted matching problem were a $O(1)$ approximation and not approximation schemes (using subexponential time). Our result resolves the status of weighted matching as well as the generalization to b–Matching.

9.3.1 Proof of Lemma 9.3.8: A Filtering Step to Bound Vertex Width

As mentioned earlier, given a graph G which has a maximum weighted b–Matching of weight $\mathcal{M}(G)$, we generate a graph G_1 such that estimating $\mathcal{M}(G')$ will give us a $(1-\epsilon)$ approximation for $\mathcal{M}(G)$. We will perform and discretization and scaling, and while it is possible to express the following in unscaled terms, we chose the scaling approach since it uses previous results in a blackbox fashion.

Bounding the maximum weight of any (useful) edge adjacent to a vertex. We now use a previous result proved in Section 6.1.3 in the context of matching

Algorithm 32 Finding G'', bounding the range of edge weights at a vertex for the Matching problem. **Except line 14**, the algorithm is the same algorithm as Algorithm 22 in Section 6.1.3

1: Let $\mathfrak{q} = 8\left\lceil \frac{1}{\epsilon^2} \log \frac{1}{\epsilon} \right\rceil$.

2: Let **tier** k to contain all edges of weight $(1+\epsilon)^k$.

3: **for each tier** $k = 0, 1, \cdots, L$ **in parallel do**

4: Find a maximal matching.

5: Let C_k be the set of nodes matched in the maximal matching.

6: Let $S_k^1 = C_k$

7: **for** $t = 1$ **to** $\mathfrak{q}$ **do**

8: Find a maximal matching between C_k and $V - S_k^t$.

9: Let T_k^t be the set of nodes matched in the maximal matching.

10: $S_k^{t+1} = S_k^t \cup T_k^t$.

11: **end for**

12: **end for**

13: Let $\varpi_i = (1+\epsilon)^k$ for the maximum k with $i \in S_k^{\mathfrak{q}}$.

14: Return $G'' = (V, E'')$ with $E'' = \{(i,j) : \frac{\epsilon^2}{\mathfrak{q}}\varpi_i, \frac{\epsilon^2}{\mathfrak{q}}\varpi_j \leq w_{ij} \leq \varpi_i, \varpi_j\}$.

and **not** b–Matching. *However we use it different way.* We first discuss the case of weighted Matching, and then show how the algorithm can be implemented for b–Matching efficiently.

Lemma 9.3.7 (Lemma 6.1.22 and 6.1.23)**.** *Given any input graph $G' = (V, E')$ which has been discretized as in Step 1, once $\{\varpi_i\}$ are computed by Algorithm 32, if we only consider the graph G^L with an edge set $\{(i,j)|(i,j) \in E_1; w_{ij} \leq \min\{\varpi_i, \varpi_j\}\}$ then $M(G') \geq M(G^L) \geq (1-3\epsilon)M(G')$ and $\sum_i \varpi_i \leq \frac{42}{\epsilon^3} \log \frac{1}{\epsilon} M(G^L)$. Here $M(G^L), M(G')$ denote the weight of the respective maximum matching in the two graphs (and* **not** *the maximum b–Matching).*

Intuitively this lemma tells us that if we find sufficiently many matchings and a particular edge $e = (i,j)$ did not appear then for one of the vertices adjacent to the edge, say j, there are many edges of similar weight. Therefore even though this edge e may have a large weight among all edges incident to i, this edge is not critical for a large matching and can be dropped. In Section 6.1.3 this was used to **upper bound** the growth of dual variables in the bipartite formulation of the matching problem. And an upper bound was sufficient for the purposes therein.

Here, we also need the **lower bound** (and that would be clear in the next step). We discard all edges incident to vertex i which has a weight less than $\epsilon^2 \varpi_i / \mathfrak{q}$. The reason is that the only handle on $M(G^L)$ we have so far is $\sum_i \varpi_i$ and if we wish to allow a small approximation for $M(G^L)$ then we need consider a fairly large range of weights at each vertex. *However the important aspect is that this range parameter is*

independent of n, B or the weights. Summarizing:

Lemma 9.3.8. *Given any input graph $G' = (V, E')$ which has been discretized as in Step 1, the graph G'' computed by Algorithm 32 satisfies $M(G') \geq M(G'') \geq (1-\epsilon)M(G^L) \geq (1-3\epsilon)(1-\epsilon)M(G') > (1-4\epsilon)M(G')$ and for any pair of edges $w_{ij}, w_{ij'}$ in G'' incident to the same vertex i, $w_{ij} \leq w_{ij'}\vartheta$ where $\vartheta = \frac{8}{\epsilon^2}\left\lceil \frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\right\rceil$. For each node $\varpi_i \geq w_{ij}$ for all j and $\sum_{i\in V}\varpi_i \leq \left(\frac{42}{\epsilon^3}\log\frac{1}{\epsilon}\right)M(G'')$. Further each $w_{ij} \geq 1$.*

Implementing Algorithm 32 for b-Matching: Any maximum weighted b–matching problem instance reduces to a maximum weighted matching problem instance with B vertices as follow: for each vertex i, we construct b_i vertices $i_1, i_2, \cdots, i_{b_i}$ and for each vector (i, j), we add edges $(i_l, j_{l'})$ for all $l \in [b_i], l' \in [b_j]$. This new graph will have $O(B)$ vertices. *We will be using this reduction virtually - as a proof and we will not be running this in the algorithm.*

Now we have a maximum weighted matching instance, we can conceptually discretize the edge weights and apply Algorithm 32. Then, we can merge vertices i_l and $i_{l'}$ (which were copies of the original vertex i) into one vertex if $\varpi_{i_l} = \varpi_{i_{l'}}$ and create a maximum weighted b-matching instance. Let $b_{i(k)}$ be the number of times any copy of vertex i has been matched in Algorithm 32 at the edge weight tier k, that is with weight $(1+\epsilon)^k$. We will create a vertex $i' = i(k)$ with $b_{i'} = b_{i(k)}$. Observe that the number of vertices we can have is now $O(nL) = O(\frac{n}{\epsilon}\log B)$ — which is one vertex per tier of edge weight. Once we have defined the copies $i(k)$ for different k, we set $\varpi_{i(k)} = (1+\epsilon)^k$. Now, $\sum_{i\in V}\sum_k \varpi_{i(k)} b_{i(k)} \leq \left(\frac{42}{\epsilon^3}\log\frac{1}{\epsilon}\right)\mathcal{M}(G')$.

An Efficient Implementation: The naive implementation of this process requires $O(\frac{B}{\epsilon}\log B)$ space. However, the algorithm can be implemented implicitly, i.e., without actual construction of intermediate maximum weighted matching instance, which is what we describe next.

Observe that there is no difference between two different copies of the vertex i, say $i_k, i_{k'}$ when we match them to a vertex j_l. *Also observe that the specific identity of the edges were not important — we were only using the maximum weight tier a vertex participated in, which implies that we do not need to remember the copies of the vertices we produce!* So when we execute Algorithm 32, we always use the copy of the vertex i with the lower index; for example, we match the copy of i_1 first and then $i_2, i_3, \cdots$ etc., whenever we match any copy of i. *Therefore it suffices to only remember the highest index that is matched for each vertex i in each edge weight tier and this uses $O(nL) = O(\frac{n}{\epsilon}\log B)$ space.* However we need to compute $b_{i(k)}$. Let $h(i,k)$ be largest index we remembered for vertex i in tier k. Clearly $b_{i(L)} = h(i,L)$. Assuming we have defined $b_{i(k')}$ for all $k' > k$ we can now define $b_{i(k)} = \max\{h(i,k) - \max_{k'>k} b_{i(k')}, 0\}$. This is because if $b_{i(k')} \geq l$ for some $k' > k$ then i_l is already matched in tier k' or above; and therefore has an edge with larger edge weight than $(1+\epsilon)^k$ incident to it and therefore $\varpi_{i_l} > (1+\epsilon)^k$.

Again, once we have defined the copies $i(k)$ for different k, we set $\varpi_{i(k)} = (1+\epsilon)^k$. For each $(i,j) \in E'$ in G' if $w_{ij} = (1+\epsilon)^k]$ we allow an edge from all $i(l)$ to all $j(l')$ where $k \leq l, l' \leq k'$ where $k' = \text{argmin}_{k''}(1+\epsilon)^{k''} \geq \mathfrak{q}(1+\epsilon)^k/\epsilon^2$. It is easy to

observe that we increase the number of edges by $O(\frac{1}{\epsilon^2}\log^2\frac{1}{\epsilon})$ factor. The number of vertices in increased by a factor $O(\frac{1}{\epsilon}\log B)$. Let this new graph be $\tilde{G} = (\tilde{V}, \tilde{E})$. Note $\mathcal{M}(G') \geq \mathcal{M}(\tilde{G}) \geq (1-3\epsilon)\mathcal{M}(G')$. Summarizing we have:

Lemma 9.3.2. *Given any input graph $G' = (V, E')$ and a b-Matching instance which has been discretized as in Step 1, we can compute a graph $\tilde{G}$ and a corresponding b-Matching instance and vertex weights $\{\varpi_i\}$ such that $\mathcal{M}(G') \geq \mathcal{M}(\tilde{G}) \geq (1-4\epsilon)\mathcal{M}(G')$ and for any pair of edges $w_{ij}, w_{ij'}$ in $\tilde{G}$ incident to the same vertex i, $w_{ij} \leq w_{ij'}\vartheta$ where $\vartheta = \frac{8}{\epsilon^2}\left\lceil\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\right\rceil$. Moreover the number of vertices $\tilde{V}$ in $\tilde{G}$ is at most $O(\frac{n}{\epsilon}\log B)$ and the number of edges $\tilde{E}$ is at most $O(\frac{m}{\epsilon^2}\log^2\frac{1}{\epsilon})$. For each node $\varpi_i \geq w_{ij}$ for all j and $\sum_{i\in\tilde{V}}\varpi_i \leq 6\epsilon\vartheta\mathcal{M}(\tilde{G})$. Further each $w_{ij} \geq 1$.*

9.3.2 Proof of Lemma 9.3.3

Lemma 9.3.3. *Let $\mathfrak{O} = \{U \subseteq \tilde{V};\ ||U||_b$ is odd, $||U||_b \leq \frac{4}{\epsilon}, 3\vartheta\varpi_j \geq \epsilon^3\varpi_i$ for all $i, j \in U\}$.*

$$\begin{aligned}
\hat{\beta} = \min \quad & \textstyle\sum_i b_i y_i + \sum_{U\in\mathfrak{O}} \lfloor ||U||_b/2 \rfloor z_U \\
s.t \quad & y_i + y_j + \textstyle\sum_{U\in\mathfrak{O}; i,j\in U} z_U \geq w_{ij} && \forall (i,j) \in \tilde{E} \\
& 2y_i + \textstyle\sum_{U\in\mathfrak{O}; i\in U} z_U \leq 2\varpi_i && \forall i \in \tilde{V} \\
& y_i, z_U \geq 0 && \forall i \in \tilde{V}; \forall U \in \mathfrak{O}
\end{aligned} \tag{LP45}$$

There exists a feasible solution $\{\hat{y}_i\}, \{\hat{z}_U\}$ for LP45 such that the objective value is $\hat{\beta} \leq (1+\epsilon)\mathcal{M}(\tilde{G})$ and $L = \{U | \hat{z}_U \neq 0\}$ is a laminar family.

Proof. Using Theorem 7.4.2 in Section 7.4.1, we have an optimal solution for LP45

such that $L = \{U | z_U \neq 0\}$ is a laminar family (and not necessarily integral). Let $(\mathbf{y}^*, \mathbf{z}^*)$ be that optimal solution. Observe that $y_i^* + \sum_{U;i\in U} z_U^* \leq \varpi_i$. If $y_i^* + \sum_{U;i\in U} z_U^* > \varpi_i$, we can simply reduce y_i^* or reduce z_U^* while increasing y_j^* for $j \in U, j \neq i$.

We construct $(\hat{\mathbf{y}}, \hat{\mathbf{z}})$ that satisfies the conditions of the lemma as follows:

$$\hat{y}_i = y_i^* + \sum_{U\in\mathcal{O}_2;i\in U,U\notin\mathfrak{O}} z_U^*/2 \quad \text{and} \quad \hat{z}_U = \begin{cases} z_U^* & \text{if } U \in \mathfrak{O} \\ 0 & \text{otherwise} \end{cases}$$

This step increases the objective value by $z_U^*/2$ for every $U \notin \mathfrak{O}$. We "charge" that increase to $\text{argmax}_{i\in U}\, \varpi_i$. To bound the charge collected by a vertex i, observe that the sets $U_1, U_2, \ldots, U_l$ that caused i to be charged, intersect at i. Using the laminarity property, we can renumber the sets such that $U_1 \subset U_2 \subset \cdots \subset U_l$. Now consider the vertex $j(i)$ that causes the charge from U_l; we know $3\vartheta\varpi_{j(i)} < \varpi_i$. Since $j(i)$ is present in all of the sets that caused i to be charged, the total charge collected by i is bounded by $\frac{1}{2}\sum_{U\in\mathcal{O};j(i)\in U} z_U \leq \frac{1}{2}\varpi_{j(i)} \leq \varpi_i/(6\vartheta)$. The total charge therefore is $\left(\sum_i \varpi_i\right)/(6\vartheta) \leq \epsilon\mathcal{M}(\tilde{G})$ (by Lemma 9.3.2) and $\mathcal{M}(\tilde{G}) = \beta^*$. The lemma follows. $\square$

9.3.3 Proof of Lemma 9.3.4

Lemma 9.3.4. *Given a (compressed; i.e., specified by a streaming algorithm) solution $\{x_{ij}\}$ to the system LP46,*

$$\begin{aligned} &\textstyle\sum_{(i,j)} w_{ij}x_{ij} = \beta \\ &\textstyle\sum_j x_{ij} \le b_i && \forall i \\ &\sum_{(i,j):i,j\in U} x_{ij} \le \left\lfloor \frac{||U||_b}{2} \right\rfloor && \forall U \in \mathfrak{O} \\ &x_{ij} \ge 0 \end{aligned} \tag{LP46}$$

we can produce a solution (a transducer which produces the solution on reading the streaming input one more time) $\{\hat{x}_{ij}\}$ using $O(\epsilon^{-3})$ passes and $O(n\,\mathrm{poly}(\epsilon^{-1}, \log n))$ space such that $\sum_{(i,j)} w_{ij}\hat{x}_{ij} = (1 - O(\epsilon))\beta$ and the constraints $\sum_j x_{ij} \le b_i$ for all i; and $\sum_{(i,j):i,j\in U} x_{ij} \le \left\lfloor \frac{||U||_b}{2} \right\rfloor$ for all $U \in \mathcal{O} = \{U|||U||_b$ is odd $\}$ (not just $\mathfrak{O}$) hold, i.e., the system LP22 is satisfied. Note that this implies a feasible fractional solution for the b–Matching problem on $\tilde{G}$.

Proof. The easiest proof of the Lemma relies on Theorems 7.2.1 and 7.2.2. We rephrase the theorems here:

Theorem 9.3.9 (Theorems 7.2.1 and 7.2.2)**.** *Let $\mathcal{O}_\delta = \{U \in \mathcal{O}, ||U||_b \le 1/\delta\}$ for a graph G with n vertices, and*

$$\lambda_U = \frac{\sum_{(i,j):i,j\in U} x_{ij}}{\lfloor \frac{||U||_b}{2} \rfloor - \frac{\delta^2||U||_b^2}{4}} \qquad \lambda = \left\{ \max_{U\in\mathcal{O}_\delta} \lambda_U, \max_i \lambda_i = \frac{\sum_j x_{ij}}{(1-4\delta)b_i} \right\}$$

If $\lambda > 1 + 8\delta$ and $\delta \in (0, \frac{1}{8}]$, given $\mathbf{x} \ge \mathbf{0}$ such that $x_{ii} = 0$ for all i, we can find $L = \{U|\lambda_U \ge \lambda - \delta^3/2; U \in \mathcal{O}_\delta\}$ in $O(\frac{m'}{\delta}\ln(1/\delta) + n\,\mathrm{poly}\{\delta^{-1}, \ln n\})$ time where $m' = |\{x_{ij} \ne 0\}|$. Moreover L defines a laminar family.

In our case $\delta = \epsilon/4$ and $\mathcal{O}_\delta = \{U \subseteq \tilde{V};\ ||U||_b \text{ is odd}, ||U||_b \leq \frac{4}{\epsilon}\}$. Therefore for every set $U \in \mathcal{O}_\delta - \mathfrak{O}$ there exists $i, j \in U$ such that $3\vartheta\varpi_j < \epsilon^3\varpi_i$. For such an U, let $high(U) = \max_{i\in U} \varpi_i$ and $low(U) = \min_{j\in U} \varpi_j$. Then $\epsilon^3\ high(U) > \vartheta\ low(U)$.

Note that $\lambda \leq 1 + 32\epsilon$ we can multiply every x_{ij} by $(1-\epsilon)/(1+32\epsilon)$ and the statement of Lemma 9.3.4 will follow. Note in our case $\lambda_i \leq 1$. Moreover, by the statement of the Lemma, we only have to consider sets in $\mathcal{O}_\delta - \mathfrak{O}$. Therefore we will use the following strategy:

(a) If we apply the algorithm in Theorem 9.3.9, such that $L \neq \emptyset$ and $\lambda > 1 + 32\epsilon$; then for each $U \in L$ and $i \in U$ it must be that $\sum_{j\in U} x_{ij} \geq \epsilon$. Otherwise we cannot have $\lambda_U \geq 1 + 32\epsilon - 32\epsilon^3$. Recall that the algorithm first constructs a graph G_φ which has at most $O(n\,\mathrm{poly}(\epsilon^{-1}))$ edges (see Section 7.5). Given a transducer which produces x_{ij}, the construction of G_φ can be done in one pass and the rest of the algorithm do not require any access to the original graph. Also, binary searches to find λ_U can be replaced by $O(\mathrm{poly}(\epsilon^{-1})$ parallel executions. Therefore, the algorithm runs in one pass, $O(n\,\mathrm{poly}(\epsilon^{-1}, \log n))$ space, and $O(m\,\mathrm{poly}(\epsilon^{-1}) + n\,\mathrm{poly}(\epsilon^{-1}, \log n))$ time.

(b) For each such $U \in L$, deleting $64\epsilon^3$ edges "fractionally" which are incident to the vertex which defines $low(U)$ and some other vertex in U (such edges exist, by Step a) will cause the updated λ_u to be less that $\lambda - 32\epsilon^3$.

(c) We will charge the deletions to the edges in Step b to the set of edges which are incident to the vertex that defines $high(U)$ and some other vertex in U — by

Step a there are at least ϵ such edges (fractionally).

(d) The weight of each of the edges that are being charged in Step c is at least $high(U)/\vartheta$ (from Lemma 9.3.8) and the weight of the edges that are causing the charge is at most $low(U)$ (again from Lemma 9.3.8). Therefore the charge to a vertex is $\frac{64\epsilon^2\vartheta\ low(U)}{high(U)}$ times its contribution to $\sum_{(i,j)} w_{ij}x_{ij}$.

(e) Moreover the charge can occur $2/\epsilon$ times due the laminarity over odd-sets of maximum size $4/\epsilon$ and thus the charge can be $\frac{128\epsilon\vartheta low(U)}{high(U)}$ times its contribution to $\sum_{(i,j)} w_{ij}x_{ij}$. Note that

$$\frac{128\epsilon\vartheta\ low(U)}{high(U)} \leq 128\epsilon^4 \tag{9.3.1}$$

After the above steps are repeated we are guaranteed that λ has decreased by $32\epsilon^3$. This can happen at most $1/(32\epsilon^3)$ before we have a ratio less than $1 + 32\epsilon$ since the initial ratio is at most $3/2+\epsilon < 2$. Therefore the total charge to any edge is at most 4ϵ times its contribution to $\sum_{(i,j)} w_{ij}x_{ij}$ using Equation 9.3.1. The lemma follows. □

9.3.4 The Oracle: Proof of Lemma 9.3.6

Lemma 9.3.6. *Given an instance of LP48 we either (i) produce a solution or a (ii) feasible solution for LP46 in time and space* $O(n\,\mathrm{poly}(\log B, \epsilon^{-1}))$.

Proof. Recall the systems:

$$\begin{aligned}
&\textstyle\sum_{(i,j)} w_{ij}x_{ij} = \beta \\
&\textstyle\sum_j x_{ij} \le b_i && \forall i \\
&\sum_{(i,j):i,j\in U} x_{ij} \le \left\lfloor \frac{||U||_b}{2} \right\rfloor && \forall U \in \mathfrak{O} \\
&x_{ij} \ge 0
\end{aligned} \tag{LP46}$$

and

$$\sum_i y'_i \sum_j u^s_{ij} + \sum_{U\in\mathfrak{O}} z'_U \sum_j u^s_{ij} \ge \sum_{(i,j)\in\tilde{E}} u^s_{ij} w_{ij}$$

$$\mathcal{P}_w : \begin{cases} \textstyle\sum_i b_i y'_i + \sum_{U\in\mathfrak{O}} \lfloor ||U||_b/2 \rfloor z'_U \le \beta \\ y'_i + y'_j + \textstyle\sum_{U\in\mathfrak{O};i,j\in U} z'_U \le \rho w_{ij} \quad \forall (i,j) \in \tilde{E} \\ y'_i, z'_U \ge 0 \end{cases} \tag{LP48}$$

Let $\gamma_w = \sum_{(i,j)\in\tilde{E}} u^s_{ij} w_{ij}$. We compute $\mathcal{S} = \{i | \sum_j u^s_{ij} > \gamma_w b_i/\beta\}$ and $\Delta_{\mathcal{S}} = \sum_{i\in\mathcal{S}} \varpi_i \sum_j u^s_{ij}$. If $\Delta_{\mathcal{S}} \ge \epsilon\gamma_w/4$ then we set $y'_i = \varpi_i \frac{\gamma_w}{\Delta_{\mathcal{S}}}$ for all $i \in \mathcal{S}$ and set all other y'_i, z'_U to 0. It is easy to see that

$$\begin{aligned}
\sum_{i\in\mathcal{S}} y'_i \sum_j u^s_{ij} + \sum_{U\in\mathcal{O}_s} z'_U \sum_{i,j\in U} u^s_{ij} &= \sum_{i\in\mathcal{S}} y'_i \sum_j u^s_{ij} = \sum_{i\in\mathcal{S}} \varpi_i \frac{\gamma_w}{\Delta_{\mathcal{S}}} \sum_j u^s_{ij} \\
&= \frac{\gamma_w}{\Delta_{\mathcal{S}}} \sum_{i\in\mathcal{S}} \varpi_i \sum_j u^s_{ij} = \frac{\gamma_w}{\Delta_{\mathcal{S}}} \Delta_{\mathcal{S}} = \gamma_w \\
\sum_i y'_i b_i + \sum_{U\in\mathcal{O}_s} z'_U \lfloor ||U||_b/2 \rfloor = \sum_{i\in\mathcal{S}} y'_i b_i &\le \frac{\beta}{\gamma_w} \sum_{i\in\mathcal{S}} y'_i \sum_j u^s_{ij} = \frac{\beta}{\gamma_w}\gamma_w = \beta
\end{aligned}$$

and $y'_i \le 4\varpi_i/\epsilon$. Since any w_{ij} adjacent to i satisfies $w_{ij} \ge \varpi_i/\vartheta$ we have that $y'_i \le w_{ij}\rho$. Thus $\{y'_i\}, \{z'_U\}$ satisfy Equation LP48 and can be returned.

Therefore, in the remainder, we assume that $\Delta_{\mathcal{S}} = \sum_{i\in\mathcal{S}} \varpi_i \sum_j u^s_{ij} < \epsilon\gamma/4$. For $i \notin \mathcal{S}$ we have $\sum_j u^s_{ij} \le \gamma_w b_i/\beta$. We **differ** from the proof of Lemma 9.2.8 here.

We will use Lemma 9.2.7, but we will use Lemma 9.2.7 multiple times per oracle. Let $W = \max_i \varpi_i$. Initially we will consider the subset $V_1 = \{i | i \notin \mathcal{S}, W \geq \varpi_i \geq \epsilon^3 W/(3\vartheta)\}$ and run Lemma 9.2.7 (we describe the parameters below). Observe that any odd subset U of V_1 with $||U||_b \leq 4/\epsilon$ will be in $\mathfrak{O}$ and also will be considered in this application of Lemma 9.2.7.

If $i, j \in V_1$ set $q_{ij} = \beta(1-\epsilon/4)u^s_{ij}/\gamma_w$, otherwise $q_{ij} = 0$. Set $\hat{q}_i = b_i, C = 1$ and apply Lemma 9.2.7 with $\nu = \epsilon$ on V_1. It is easy to verify (exactly as in Proof of Lemma 9.2.8) that we satisfy **A1**, **A2**, and **A3**. Using Lemma 9.2.7 we get a collection of **disjoint** odd-sets $\mathcal{K}_1$ where each $U \in \mathcal{K}_1$ satisfies

$$\sum_{i,j \in U} \left(1 - \frac{\epsilon}{4}\right) \frac{\beta}{\gamma_w} u^s_{ij} \geq \frac{1}{2}\left(\sum_{i \in U} b_i - 1\right) = \left\lfloor \frac{||U||_b}{2} \right\rfloor \geq \frac{1}{3}||U||_b \geq \frac{1}{3}\sum_{i \in U}\sum_{j} \frac{\beta}{\gamma_w} u^s_{ij} \tag{9.3.2}$$

$$\sum_{i,j \in U} u^s_{ij} \geq \sum_{i \in U}\sum_{j \notin U} u^s_{ij} = \mathcal{C}(U, u^s) \tag{9.3.3}$$

Let $\Delta_{\mathcal{K}_1} = \sum_{U \in \mathcal{K}_1} (\max_{i \in U} \varpi_i) \sum_{i,j \in U} u^s_{ij}$. But we **do not** return the oracle immediately. Let $V(\mathcal{K}_1) = \cup_{U \in \mathcal{K}_1} U$. We now consider all odd sets in V_2 which would consider $i : \frac{\epsilon^3 W}{3\vartheta(1+\epsilon)} \leq \varpi_i \leq \frac{W}{(1+\epsilon)}$; but we exclude vertices in $\mathcal{S}$ and V_1 (recall edge weights are powers of $(1+\epsilon)$, Lemma 9.3.2). In general we would consider V_ℓ where

$$V_\ell = \left\{ i \,\middle|\, i \notin \mathcal{S}, i \notin \bigcup_{l=1}^{\ell-1} V_l, \frac{\epsilon^3 W}{3\vartheta(1+\epsilon)^{\ell-1}} \leq \varpi_i \leq \frac{W}{(1+\epsilon)^{\ell-1}} \right\}$$

And we would find a set of odd-sets $\mathcal{K}_\ell$ where each $U \in \mathcal{K}_\ell$ also satisfy Equations 9.3.2 and 9.3.3. We define $\Delta_{\mathcal{K}_\ell} = \sum_{U \in \mathcal{K}_\ell} (\max_{i \in U} \varpi_i) \sum_{i,j \in U} u^s_{ij}$.

We will run Lemma 9.2.7 $O(\frac{1}{\epsilon}\log B)$ times (recall that is the number of distinct

weights by construction (from Lemma 9.3.2). If $\sum_\ell \Delta_{\mathcal{K}_\ell} \geq \epsilon\gamma_w/8$ then for $U \in \mathcal{K}_\ell$ we set

$$z'_U = \frac{\gamma_w \left(\max_{i\in U} \varpi_i\right)}{\sum_\ell \Delta_{\mathcal{K}_\ell}} \leq \frac{8}{\epsilon}\left(\max_{i\in U} \varpi_i\right)$$

All $y'_i = 0$ and for any $U \in \mathfrak{O}$ which is not in any $\mathcal{K}_\ell$ set $z'_U = 0$. Now;

$$\begin{aligned}\sum_i y'_i \sum_j u_{ij} + \sum_{U\in\mathfrak{O}} z'_U \sum_{i,j\in U} u_{ij} &= \sum_{U\in\mathfrak{O}} z'_U \sum_{i,j\in U} u_{ij} \\ &= \sum_\ell \frac{\gamma_w \left(\max_{i\in U} \varpi_i\right)}{\sum_{\ell'} \Delta_{\mathcal{K}_{\ell'}}} \sum_{i,j\in U} u_{ij} = \frac{\gamma_w \sum_\ell \Delta_{\mathcal{K}_\ell}}{\sum_{\ell'} \Delta_{\mathcal{K}_{\ell'}}} = \gamma_w\end{aligned}$$

$$\begin{aligned}\sum_i y'_i b_i + \sum_{U\in\mathfrak{O}} z'_U \left\lfloor \frac{||U||_b}{2} \right\rfloor &= \sum_{U\in\mathfrak{O}} z'_U \left\lfloor \frac{||U||_b}{2} \right\rfloor \\ &\leq \sum_{U\in\mathfrak{O}} z'_U \frac{\beta}{\gamma_w} \sum_{i,j\in U} u_{ij} \quad \text{(Using Equation 9.3.2)} \\ &= \frac{\beta}{\gamma_w} \sum_{U\in\mathfrak{O}} z'_U \sum_{i,j\in U} u_{ij} = \frac{\beta}{\gamma_w}\gamma_w = \beta\end{aligned}$$

Observe that $\sum_{i\in U; U\in\mathfrak{O}} z'_U \leq 24\vartheta\varpi_i/\epsilon$ due to disjointedness of the collection $\cup_\ell \mathcal{K}_\ell$ (since the vertex sets were separate) and the fact that for any $i \in U; U \in \mathfrak{O}$ we have $3\vartheta\varpi_i \leq \epsilon^3 \left(\max_{i\in U} \varpi_i\right)$. Now for any w_{ij} adjacent to i we have $\varpi_i \leq \vartheta w_{ij}$. Therefore $\sum_{i,j\in U; U\in\mathfrak{O}} z'_U \leq (24\vartheta^2/\epsilon^4) w_{ij} = \rho w_{ij}$. Therefore $\{y'_i\}, \{z'_U\}$ satisfy the system LP48 and can be returned (note all sets satisfy Equation 9.3.3).

Thus the only case we are left to discuss is $\sum_\ell \Delta_{\mathcal{K}_\ell} < \epsilon\gamma_w/8$ and $\Delta_{\mathcal{S}} \leq \epsilon\gamma_w/4$. We show that $\hat{\beta} \geq (1-2\epsilon)\beta$. Let $V_{bad} = \mathcal{S} \cup \{i | i \in U; U \in \mathcal{K}_\ell\}$; the set of vertices associated with any $\mathcal{K}_\ell$. Suppose $U \subseteq \tilde{V} \setminus V_{bad}$ be an odd-set in $\mathfrak{O}$. Then we would have considered this set in one of the invocations of Lemma 9.2.7. Since $U \notin \mathcal{K}_\ell$ and

$U \in \mathfrak{O}$, we have

$$\sum_{i,j \in U} \left(1 - \frac{\epsilon}{4}\right) \frac{\beta}{\gamma_w} u^s_{ij} \leq \left\lfloor \frac{||U||_b}{2} \right\rfloor + \epsilon/2 = \left(1 + \frac{\epsilon}{2}\right) \left\lfloor \frac{||U||_b}{2} \right\rfloor$$

Moreover for $i \in \tilde{V} \setminus V_{bad}$ we have $\sum_j \frac{\beta}{\gamma_w} u^s_{ij} \leq b_i$. Therefore setting $x_{ij} = \frac{\beta}{\gamma_w} \frac{(1-\epsilon/4)}{(1+\epsilon/2)} u_{ij}$ for $i, j \in \tilde{V} \setminus V_{bad}$ and $x_{ij} = 0$ otherwise. We will satisfy the constraints of LP46 — observe the vertices in V_{bad} do not matter since all edges incident to them have been "deleted". Therefore all that remains is to evaluate $\sum_{i,j} x_{ij}$, which is given by:

$$\begin{aligned}
\frac{(1+\epsilon/2)}{(1-\epsilon/4)} \frac{\gamma_w}{\beta} \sum_{i,j} w_{ij} x_{ij} &= \frac{(1+\epsilon/2)}{(1-\epsilon/4)} \frac{\gamma_w}{\beta} \sum_{i,j \in \tilde{V} \setminus V_{bad}} w_{ij} x_{ij} \\
&= \sum_{i,j \in V \setminus V_{bad}} w_{ij} u^s_{ij} = \sum_{i,j} w_{ij} u^s_{ij} - \sum_{i \in V_{bad}} \sum_j w_{ij} u^s_{ij} \\
&= \gamma_w - \sum_{i \in \mathcal{S}} \sum_j w_{ij} u^s_{ij} - \sum_\ell \sum_{U \in V(\mathcal{K}_\ell)} \sum_{i \in U} \sum_j w_{ij} u^s_{ij} \\
&\qquad \text{(All } V(\mathcal{K}_\ell), \mathcal{S} \text{ are mutually disjoint)} \\
&\geq \gamma_w - \sum_{i \in \mathcal{S}} \sum_j \varpi_i u^s_{ij} - \sum_\ell \sum_{U \in V(\mathcal{K}_\ell)} \sum_{i \in U} \sum_j \varpi_i u^s_{ij} \qquad \text{(Definition of } \varpi_i) \\
&\geq \gamma_w - \sum_{i \in \mathcal{S}} \varpi_i \sum_j u^s_{ij} - \sum_\ell \sum_{U \in V(\mathcal{K}_\ell)} \left(\max_{i \in U} \varpi_i\right) \sum_{i \in U} \sum_j u^s_{ij} \\
&\geq \gamma_w - \Delta_{\mathcal{S}} - \sum_\ell \sum_{U \in V(\mathcal{K}_\ell)} \left(\max_{i \in U} \varpi_i\right) 3 \sum_{i,j \in U} u^s_{ij} \quad \text{(Using Equation 9.3.2)} \\
&= \gamma_w - \Delta_{\mathcal{S}} - 3 \sum_\ell \Delta_{\mathcal{K}_\ell} \leq \gamma_w - \epsilon\gamma_w/4 - 3\epsilon\gamma_w/8 \geq \left(1 - \frac{5\epsilon}{8}\right) \gamma_w
\end{aligned}$$

Therefore $\sum_{i,j} x_{ij} \geq (1 - 5\epsilon/8)(1 - \epsilon/4)\beta/(1 + \epsilon/2) \geq (1 - 3\epsilon/2)\beta$. This proves that we have a feasible solution for LP46. This concludes the proof of the oracle. $\square$

9.4 Deferred Sparsification

Definition 9.4.1 (The Deferred Sparsifier Problem)**.** Consider the problem: We are given a weighted graph $G = (V, E, u)$ but the weights are **not revealed** to us. Instead we are given $\{\varsigma_{ij}\}$ with the promise that $\varsigma_{ij}/\chi \leq u_{ij} \leq \varsigma_{ij}\chi$. We can have a single pass over the $m = |E|$ edges and have to produce a smaller summary data structure $\mathcal{D}$ storing only some of the edges. Once the data structure is constructed, **then** the exact weights u_{ij} of only those edges stored in our structure $\mathcal{D}$ are revealed to us. We then have to output a sparsifier $H = (V, E', u^s)$. **Note that this is implemented in a single pass over the graph.**

Lemma 9.4.2. *Given bound ϱ and $\{\varsigma_{ij}\}$ factor and edge weights are in $[1, O(\operatorname{poly} n)]$, we can generate a deferred sparsification of size $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$. We can construct a $(1 \pm \xi)$-sparsification from the deferred sparsification when the edge weights of the stored edges are revealed. The algorithm runs in $O(m(\log\chi + \log\frac{1}{\xi} + \log\log n)\alpha_{m,n})$ time and $O(\frac{\chi^2 n}{\xi^2}\log^4 n)$ space. In this paper $\chi = n^{1/(4p)}$.*

Proof. Recall the semi-streaming algorithm for the graph sparsification (see Algorithm 14). The main intuition (which in some form dates back to [19]) we need is that weighted sparsifiers are constructed using an *unweighted graph* by (i) we first determine the probability p_e of sampling the edge e and then (ii) if e is sampled then assign it a value/importance w_e/p_e. We can simply decouple these two steps in a deferred sparsifier. For a weighted graph with weights in $[1, \chi^2]$ the the number of sampled edges increase by χ^2. For a graph with larger deviation of edge weights, this

process is repeated over a layered graph where the edge weights differ by powers of Λ and this process runs independently for each layer. We can easily observe that the sum (allowing multiple edges between two nodes) of sparsifiers of a set of graphs is a sparsifier of the sum of the graphs. We can split the graph using ς_{ij} values, construct the deferred sparsifier of each graph, and then take the sum of the sparsifiers. The Lemma follows. $\square$

9.5 Disentangling Compressed Fractional Solutions

In this section we prove the following Theorem:

Theorem 9.5.1. *Given a streaming oracle that provides x_{ij} for the edges (i,j) in the formulation LP22, such that $\sum_{(i,j)} w_{ij}x_{ij} \geq (1-\varepsilon)\beta^*$, we can compute an integral $(1-5\varepsilon)$ approximate solution using a single additional pass, time $O(n\varepsilon^{-6}(\log \frac{1}{\varepsilon})\log n)$ and space $O(n\varepsilon^{-5}\log n)$. The exponents of ε are slightly better for the bipartite case. Recall that:*

$$\begin{aligned} \beta^* = \quad & \max \textstyle\sum_{(i,j)} w_{ij}x_{ij} \\ & \textstyle\sum_j x_{ij} \leq b_i && \forall i \in V \\ & \textstyle\sum_{(i,j):i,j\in U} x_{ij} \leq \lfloor ||U||_b/2 \rfloor && \forall U \in \mathcal{O} \\ & x_{ij} \geq 0 \end{aligned} \tag{LP22}$$

We are given an access to a data structure that on being provided $(i,j) \in E$ gives us a fractional assignment y_{ij} such that we satisfy the relaxed b–Matching polytope (where we only have constraints for each odd set U such that $\sum_{i\in U} b_i = ||U||_b \leq 4/\varepsilon$). We assume that $y_{ij} = 0$ if $(i,j) \notin E$.

Definition 9.5.2. For $(i,j) \in E$; if $b_i < b_j$ then let $h(i,j) = i$ else $h(i,j) = j$.

Definition 9.5.3. Let $r = 100\varepsilon^{-3}\log n$. For a vertex i let $\textsc{High}(i) = \{(i,j)|(i,j) \in E, x_{ij} \geq b_i/r\}$. Let $\textsc{Low}(i) = \{(i,j)|(i,j) \in E, x_{ij} \leq b_i/r\}$. Note that $\sum_j x_{ij} \leq b_i$ then $|\textsc{High}(i)| \leq r$.

Fact 9.5.4. **(Bernstein Inequality)** Given a sequence of independent Bernoulli random variables $\{X_i\}$ with respective expectations and variances $\{\mathrm{E}[X_i], Var[X_i]\}$, such that for all i, $X_i \leq E[X_i] + M$ we have

$$\Pr\left[\sum_i X_i - \sum_i \mathrm{E}[X_i] \geq t\right] \leq \exp\left(-\frac{t^2}{\sum_i Var[X_i] + 2tM/3}\right)$$

Lemma 9.5.5. *Assuming $\beta = \sum_{(i,j)} x_{ij}w_{ij} \geq (1-\varepsilon)\beta^*$ where β^* is the optimum b–Matching, the ensuing algorithm* OneStep *provides a feasible fractional $(1-2\varepsilon)$ optimum b–Matching with high probability with $O(rn)$ non-zero edges.*

Algorithm OneStep*:*

(1) For every vertex i we remember the edges in $\textsc{High}(i)$ along with the and the assignment x_{ij}. For any edge (i,j) such that it is in $\textsc{High}(i)$ or $\textsc{High}(j)$ we set $Y_{ij} = x_{ij}$.

(2) For every other $(i,j) \in E$: Let $X_{ij} = \begin{cases} \frac{b_{h(i,j)}}{r} & \textit{With probability } \frac{rx_{ij}}{b_{h(i,j)}} \\ 0 & \textit{Otherwise} \end{cases}$.

(3) Set $\hat{x}_{ij} = X_{ij}/(1+\varepsilon)^2$.

Proof. First let us consider the objective function. Let $\beta = \sum_{(i,j)} x_{ij}w_{ij}$. Let $Z_{ij} = \left(\frac{b_{h(i,j)}}{r} - X_{ij}\right) w_{ij}$. Now it is straightforward to see that $\beta^* \geq b_{h(i,j)}w_{ij}$ for all edges (i,j) — consider a b–matching with just this edge. As a consequence $Z_{ij} - EZ_{ij} \leq \beta^*/r$

and

$$\sum_{(i,j)} Var[Z_{ij}] \leq \sum_{(i,j)} \frac{w_{ij}^2 b_{h(i,j)}^2}{r^2} \frac{r x_{ij}}{b_{h(i,j)}} \leq \frac{1}{r} \sum_{(i,j)} \left(w_{ij} b_{h(i,j)}\right) x_{ij} w_{ij} \leq \frac{\beta^* \beta}{r}$$

And therefore:

$$\Pr\left[\sum_{(i,j)} Z_{ij} \leq \sum_{(i,j)} \mathrm{E}\left[Z_{ij}\right] - \varepsilon\beta^*\right] \leq \exp\left(-\frac{\varepsilon^2 r}{\frac{\beta}{\beta^*} + 2\varepsilon/3}\right)$$

or in other words the objective function is preserved with high probability if $\beta \geq \beta^*/2$. Now for each constraint at vertex i, for each (i,j) we have $X_{ij} - \mathrm{E}\left[X_{ij}\right] \leq \frac{b_{h(i,j)}}{r} \leq \frac{b_i}{r}$ and

$$Var[X_{ij}] \leq \frac{b_{h(i,j)}^2}{r^2} \frac{r x_{ij}}{b_{h(i,j)}} = \frac{b_{h(i,j)} x_{ij}}{r} \leq \frac{b_i x_{ij}}{r}$$

As a consequence $\sum_j Var[X_{ij}] \leq \frac{b_i}{r} \sum_j x_{ij} \leq \frac{b_i^2}{r}$. Thus using Bernstein Inequality, Fact 9.5.4 we get

$$\Pr\left[\sum_{(i,j)} X_{ij} \geq (1+\varepsilon) b_i\right] \leq \exp\left(-\frac{\varepsilon^2 r}{1 + 2\varepsilon/3}\right)$$

Or in other words the vertex constraints are satisfied with high probability. If we were only considering bipartite graphs, we can stop at this point — note that $r = \Omega(\varepsilon^{-2} \log n)$ suffices in that case.

Observe that this same reasoning extends to the odd set constraints of size $4/\varepsilon$ or less. There are potentially $n^{4/\varepsilon}$ such odd sets. This is the reason we need $r\varepsilon^2 = \Omega((\log n)/\varepsilon)$. It is easy to see that if $X_{ij}/(1+\varepsilon)$ satisfies all vertex constraints then $X_{ij}/(1+\varepsilon)^2$ satisfies all odd set constraints where the odd-set has size $\sum_i b_i \geq 4/\varepsilon$. Therefore $X_{ij}/(1+\varepsilon)^2$ is a feasible solution for the b–Matching LP. The Lemma follows. □

We now use Theorem 7.1.4 and Theorem 9.5.1 follows.

Chapter 10

Application IV: Multicut and Correlation Clustering

Chapter Outline: In this chapter, we present one-pass algorithms for the multicut problem and the correlation clustering problem. The algorithms in this chapter combine the sparsification algorithm and existing algorithms. The existing algorithms use LPs for the multicut problem and correlation clustering (minimization) and an SDP for correlation clustering (maximization). We apply primal-dual algorithms in order to obtain space- and time-efficient algorithms.

10.1 Problem Definitions and Techniques

The multicut problem is relatively well-studied problem and is defined as follows:

Definition 10.1.1 (The Multicut Problem)**.** Given a weighted undirected graph and

k pairs of vertices (s_i, t_i), the multicut problem is to minimize the weight of the subset of edges whose removal disconnects these pairs.

$\Omega(\log n)$ lower bound is known for the multicut problem [32] and a matching $O(\log n)$-approximation algorithm is also known [52]. The algorithm uses a region growing technique [72].

Definition 10.1.2 (Correlation Clustering)**.** In the correlation clustering problem, we are given a graph G with each edge annotated either '+' (positive) or '-' (negative). We denote the set of positive edges as E^+ and the set of negative edges as E^-. A clustering $C = \{C_1, C_2, \cdots, C_\ell\}$ *agrees* with (u, v) if either (a) $(u, v) \in E^+$ and $u, v \in C_i$ for some i, or (b) $(u, v) \in E^-$ and $u \in C_i, v \in C_j$ for some $i \neq j$. Otherwise, we say that C *disagrees* with (u, v). The objective of the correlation clustering problem is either (i) maximize the agreements or (ii) to minimize the disagreements. These two problems are equivalent for the optimal solution, but they are different if we consider approximation algorithms.

Correlation clustering problem was proposed by Bansal et al. [14]. They present algorithms for maximizing agreements and minimizing disagreements in complete unweighted graphs. Demaine et al. [33] observe the connection between the minimization version of correlation clustering and the multicut problem. They also presented an $O(\log n)$-approximation algorithm for the minimization version using the region growing technique. There has a number of subsequent results on correlation clustering, for example, [91, 25, 9] but the space requirement of the algorithms (in the

general case) has not been addressed. Note that the number of clusters is *not* fixed in advance and arises from the optimization naturally and therefore expressing this as a CSP requires an alphabet of size $O(n)$. Although there is a time- and space-efficient algorithm for CSP (constraint satisfaction problem) [90], the algorithm's running time and space requirement depends exponentially on the alphabet size. Also note that some of the previous results are efficient for special cases of edge weights (namely when the weights satisfy a triangle inequality, or the graph is a complete graph) — we do not make any such assumption. Especially, the algorithms for general weighted graphs solve LP and SDP [33, 91, 25].

Techniques: We first sparsify the input graph so that we can remember the whole graph in $\tilde{O}(n)$ space which will be explained. We then construct and approximate an LP or an SDP for the sparsified graph. We use the multiplicative weights update method ([11], see Section 2.3.1) for approximating the LP and use the SDP feasibility framework ([12, 90], see Section 2.3.3) for approximating the SDP.

For the multicut problem and the minimization version of correlation clustering, we introduce a new idea for constructing an oracle. The LPs for the problems have either (i) triangular inequality or (ii) exponentially many constraints. However note that we are interested in the final rounding of the fractional solution rather than the feasibility of the fractional solution. Therefore, we round the fractional solution using existing rounding algorithms [52, 33]. If it fails to produce an integral solution, we find a constraint which causes the failure and produce a primal witness using the

constraint. This method can be viewed as a "simpler" variant of the dual-primal approach.

For the maximization version of correlation clustering, the best known approximation algorithm for uses an SDP formulation [91]. We develop a semi-streaming algorithm using the explicit verification approach.

10.2 The Multicut Problem

In this section, we present a $O(\log n)$-approximation algorithm for the multicut problem. The multicut problem can be solved using LP-rounding approach. We solve the LP in the semi-streaming model and round the fractional solution using the region growing technique. We proceeds as follows.

1. We sparsify the graph. The sparsification algorithm preserves the every multicut value within $1 \pm \delta$ factor. In the rest of the algorithm, we use this sparsification as our input graph and therefore, we have a single-pass algorithm.

2. We repeatedly guess the optimal solution α using a binary search. For each guess α,

   (a) Given α, we contract all edges with weight greater than α and remove all edges with weight less than $\delta\alpha/m'$ where m' is the number of edges in the sparsification. The contracted edges have to be satisfied in order for α to be a correct guess. Removed edges contribute at most $\delta\alpha$ to the objective

value and hence, can be ignored.

(b) We reformulate the linear program slightly. The LP formulation will be explained in Section 10.2.1. and we solve the LP with the multiplicative weights update method. The oracle consists of two phases. It first rounds the current fractional solution. The rounding algorithm also checks if there exists a violated constraint. If there is such a constraint, the oracle constructs a dual witness with the violated constraint. Otherwise, the rounding algorithm produces a multicut. The oracle will be presented in Section 10.2.2.

Summarizing the above, we obtain (see Section 10.2.2 for the proof):

Theorem 10.2.1. *There exists a single-pass $O(\log k)$-approximation algorithm for the multicut problem in the semi-streaming model that runs in $\tilde{O}(\frac{n^2}{\epsilon^7})$ time and $\tilde{O}(\frac{n}{\epsilon^2} + k)$ space.*

10.2.1 Linear Programming Formulation

A multicut problem can be formulated as LP50. In LP50, x_{ij} indicates if (i, j) is in the multicut or not. If $x_{ij} = 1$, (i, j) is in the multicut and otherwise, it is not in the multicut. We now interpret the x_{ij} as assignment of lengths and write the following LP:

$$\begin{aligned} \min \quad & \textstyle\sum_{(i,j)\in E} w_{ij} x_{ij} \\ \text{s.t} \quad & \textstyle\sum_{(k,l)\in p} x_{kl} \geq 1 \quad \text{for all } s_i - t_i \text{ path } p \end{aligned} \tag{LP50}$$

We replace x_{ij} by $\frac{1}{w_{ij}} x'_{ij}$. We obtain an equivalent LP but it is easier to explain the oracle when we have x'_{ij} instead of x_{ij}. LP51 and LP52 are the resulting LP and its dual LP.

$$\begin{aligned} \min \quad & \textstyle\sum_{(i,j)\in E} x'_{ij} \\ \text{s.t} \quad & \textstyle\sum_{(k,l)\in p} \frac{1}{w_{kl}} x'_{kl} \geq 1 \quad \text{for all } s_i - t_i \text{ path } p \end{aligned} \tag{LP51}$$

$$\begin{aligned} \max \quad & \textstyle\sum_{p} \beta_p \\ \text{s.t} \quad & \textstyle\frac{1}{w_{ij}} \sum_{(i,j)\in p} \beta_p \leq 1 \quad \text{for all } (i,j) \in E \end{aligned} \tag{LP52}$$

10.2.2 Oracle

We start with LP52 since it contains only $\tilde{O}(n\epsilon^{-2})$ constraints and therefore, require a smaller number of iterations given the same width parameter. Now the framework maintains a fractional solution of LP51 and the oracle tries to round it. The rounding algorithm for the multicut problem uses the region growing technique [72, 52, 92]. For the details of the technique and proof, see Chapter 20 of [92].

Definition 10.2.2. Let l_{uv} be the edge length of an edge (u, v) and let $d(u, v)$ be the distance, i.e., the length of the shortest $u - v$ path. Given edge lengths, a ball $B(u, r)$

is defined as $\{v|d(u,v) \leq r\}$. We define $c(B(u,r))$ as the total weight of edges that are cut by $B(u,r)$, i.e., (v_1, v_2) such that $d(u,v_1) \leq r < d(u,v_2)$.

Also let $vol(B(u,r))$ be the volume of $B(u,r)$ which is defined as the sum of $l_{v_1v_2}w_{v_1v_2}$ for $(v_1,v_2) \in B(u,r)$. For edge (v_1,v_2) with $d(u,v_1) \leq r < d(u,v_2)$, we assume that $B(u,r)$ contains $\frac{r-d(u,v_1)}{l_{v_1v_2}}$ fraction of the edge volume. We denote the total edge volume as F and for the sake of the proof, we assume that $vol(B(u,0)) = F/k \neq 0$.

The region growing process starts at an arbitrary node u and the radius r of the ball gradually increases until $c(B(u,r)) = (C \log k)vol(B(u,r))$. The following lemma bounds the radius of $B(u,r)$.

Lemma 10.2.3. *[52, 92, 33] If $C = 3\ln(k+1)$, the ball stops growing before the radius becomes $1/3$.*

The rounding algorithm repeatedly applies the region growing technique with x_{ij} as edge lengths and declares each ball as a cluster. Since the radius of each ball is at most $1/3$ and the distance between any source-sink pair is at least 1, the rounding algorithm returns a feasible multicut.

Since the constructed balls are disjoint, the total volume is bounded by twice of the LP solution (due to the edge volume and volume on each node). And the total cut value is bounded by $O(\log k)$ times the total volume and therefore, the cut value of the rounded solution is bounded by $O(\log n)$ times the LP solution.

Note that the algorithm actually finds a (shortest) path from u to v if v belongs to

Algorithm 33 Oracle for LP51

1: Normalize x_{ij} so that $\sum_{ij} w_{ij}x_{ij} + \sum_{ij} w_{ij}y_{ij} = \alpha$.

2: Let $l_{ij} = x_{ij}$ for $(i,j) \in E^+$ and $l_{ij} = 1 - x_{ij}$ for $(i,j) \in E^-$.

3: Round the fractional solution.

4: **if** u and v belong to the same cluster for $(u,v) \in E^-$ and $l_{uv} > 2/3$ **then**

5: Find the corresponding path p (that violates a constraint).

6: Return $\beta_p = \alpha$ and $\beta_{p'} = 0$ for $p' \neq p$.

7: **else**

8: Return the rounded solution.

9: **end if**

the ball starting at u. If the triangle inequality is violated and the rounding algorithm puts a source-sink pair in the same cluster, we can actually find the violated constraint and construct a dual witness with the violated constraint. If it does not find any such source-sink pair, the rounding algorithm returns a valid multicut and the objective value is at most $O(\log n)$ times the LP solution. We use this fact for the oracle.

The next lemma is straightforward:

Lemma 10.2.4. *The oracle for LP51 returns an admissible dual witness with the width parameter m'/δ where m' is the number of edges in the sparsification.*

Proof. By the way its constructed and Lemma 6.1.1, the oracle returns an admissible dual witness. By the construction, $\beta_p = \alpha$ and only one β_p has a non-zero value. Since we remove all the edges of weight less than $\delta\alpha/m'$, the width parameter is bounded

by $\frac{\alpha m'}{\delta\alpha} = m'/\delta$. $\square$

Proof of Theorem 10.2.1. By Theorem 2.3.3 and the width parameter, the multiplicative weights update method finds an (almost) feasible solution of LP52 in $\tilde{O}(m'/\delta^3) = \tilde{O}(n/\delta^5)$ iterations. Each iteration takes $\tilde{O}(m')$ time and therefore, we find either a rounded solution (with value at most $O(\log n)\alpha$) or a (almost) feasible dual solution (which proves that our guess α is smaller than or close to the optimal solution) in $\tilde{O}(\frac{n^2}{\epsilon^7})$ time. The binary search for α requires $O(\log m')$ repetitions. Summarizing, we obtain the desired result. $\square$

10.3 Correlation Clustering: Minimizing Disagreements

In this section, we present a $O(\log n)$-approximation algorithm for the minimization correlation clustering problem with an additive error $O(\delta W)$ where W is the total edge weight. If the optimal disagreement is more than $\delta W/\log n$, the algorithm returns a $O(\log n)$-approximation solution. On the other hand, if it is smaller than $\delta W/\log n$, it returns a solution with a $O(\delta W)$ additive error. We denote this as a $(O(\log n), O(\delta W))$-approximation. Observe that this is different than using the solution for the maximization version. Since the maximization is an $O(1)$ approximation, using that algorithm will not allow us to get arbitrarily small additive error as is achieved by our algorithm.

Our algorithm is based on the $O(\log n)$-approximation algorithm given in [33]. The basic idea is to apply the algorithm to the semi-streaming model. However, the algorithm solves a linear program with $O(n^2)$ variables and $O(n^3)$ constraints which results in several complications when applied to the semi-streaming model. We use the intuition derived from multicut, namely that, if there are few edges which are negative then we can solve the correlation clustering using an algorithm similar to the Multicut algorithm. However there are some subtle but significant differences, namely, in the correlation clustering model we need not separate a negative edge. This corresponds to an "outlier version" (a Lagrangian relaxation) of the multicut problem where there is a penalty for not cutting an edge. In the following we assume that we have $\omega(n \operatorname{poly} \log n)$ negative edges. If the number of negative edges is $O(n \operatorname{poly} \log n)$ then we can omit step 1 and eliminate the additive guarantee. We proceed as follows.

1. We sparsify the graph. The sparsification algorithm preserves the optimal solution within $\frac{\delta W}{\log n}$ additive error.

2. We reformulate the linear program so that it has $\tilde{O}(n)$ variables but exponentially many constraints. Limiting the number of variables enables us to approximate the linear program in near linear space. The LP formulation will be explained in Section 10.3.1.

3. We solve the linear program with the multiplicative weights update method. The oracle consists of two phases. It first rounds the current fractional solution. Rounding algorithm also checks if there exists a violated constraint. If there

is one, the oracle constructs a dual witness with the violated constraint. The oracle will be presented in Section 10.3.2.

4. If the rounding algorithm succeeds, it returns a $(O(\log n), O(\delta W))$-approximation solution.

Summarizing the above, we obtain:

Theorem 10.3.1. *There is an one-pass* $(O(\log n), O(\epsilon W))$*-approximation algorithm for the minimizing disagreements problem in the semi-streaming model that runs in* $\tilde{O}(\frac{n^2}{\epsilon^7})$ *time.*

Proof. From the width parameter, the number of iterations is $\tilde{O}(n/\delta^5)$ and the running time for each iteration is $\tilde{O}(n/\delta^2)$. Therefore, the running time of the algorithm is $\tilde{O}(n^2/\delta^7)$.

The optimal fractional solution for the sparsification is bounded by $OPT + \frac{2\delta W}{\log n}$. The rounded solution has at most $O(\log n)$ times the fractional solution which is at most $O(\log n)OPT + O(\delta)W$. Ignored edges (due to small edge weight) and the sparsification process contributes additional δW and $\frac{2\delta W}{\log n}$ additive error. By summing up all the errors, we obtain the desired result. □

10.3.1 Linear Program Formulation

LP53 is the LP used in [33]. The LP is similar to LP50 except there are negative edges and the edge length is bounded by 1 due to the objective value of negative

edge length. If there is no upperbound on the edge length, we can set the length of a negative edge to be infinite to obtain a negative infinite objective value.

$$\begin{aligned} \min \quad & \textstyle\sum_{(i,j)\in E^-} w_{ij}(1-x_{ij}) + \sum_{(i,j)\in E^+} w_{ij}x_{ij} \\ \text{s.t} \quad & x_{ik} + x_{kj} \le x_{ij} \quad \text{for all } i,j,k \in V \\ & x_{ij} = x_{ji} \qquad\qquad \text{for all } i,j \in V \\ & 0 \le x_{ij} \le 1 \qquad\quad \text{for all } i,j \in V \end{aligned} \tag{LP53}$$

As in Section 10.2, we reformulate the LP so that the constraints correspond to paths and there are m' variables in the LP. We also replace x_{ij} by $1-y_{ij}$ for negative edges. Then, again we change variables x_{ij} and y_{ij} by $x'_{ij} = w_{ij}x_{ij}$ and $y'_{ij} = w_{ij}y_{ij}$. We obtain LP54 and LP55.

$$\begin{aligned} \min \quad & \textstyle\sum_{(i,j)\in E^-} y'_{ij} + \sum_{(i,j)\in E^+} x'_{ij} \\ \text{s.t} \quad & \tfrac{1}{w_{ij}} y'_{ij} + \textstyle\sum_{(k,l)\in p} \tfrac{1}{w_{kl}} x'_{kl} \ge 1 \quad \text{for all } p \in P^+_{ij}, (i,j) \in E^- \end{aligned} \tag{LP54}$$

$$\begin{aligned} \max \quad & \textstyle\sum_p \beta_p \\ \text{s.t} \quad & \tfrac{1}{w_{ij}} \textstyle\sum_{p:i-j \text{ path}} \beta_p \le 1 \quad \text{for all } (i,j) \in E^- \\ & \tfrac{1}{w_{ij}} \textstyle\sum_{(i,j)\in p} \beta_p \le 1 \qquad\quad \text{for all } (i,j) \in E^+ \end{aligned} \tag{LP55}$$

10.3.2 Oracle

As in Section 10.2, we start with LP55 rather than LP54 and the oracle uses the rounding algorithm in [33] which is also based on the region growing technique. The

rounding algorithm first applies to the region growing technique on positive edges. Let LP^+ and LP^- be the objective values induced by the positive edges and the negative edges respectively and let $l_{v_1v_2} = x_{v_1v_2}$. By Lemma 10.2.3 guarantees that the disagreements with positive edges is bounded by $O(\log n)LP^+$ while radius of each ball is bounded by 1/3.

Now consider negative edges. In order for a negative edge (v_1, v_2) to disagree with the clustering, $d(u, v_1), d(u, v_2) < 1/3$ for some u. By the triangular inequalities, $l_{v_1v_2} < 2/3$ which means that the disagreement with (v_1, v_2) in the fractional solution is at least 1/3. Therefore, the disagreements with negative edges is bounded by $3LP^-$.

Note that the algorithm actually finds a path from u to v_1 and v_2. If the triangular inequality is violated and the rounding algorithm puts v_1 and v_2 in the same cluster for $x_{v_1v_2} > 2/3$, we can actually find the violated constraint and construct a dual witness with the violated constraint.

Again, the next lemma is straightforward and using it Theorem 10.3.1 follows.

Lemma 10.3.2. *The oracle for LP55 returns an admissible dual witness with the width parameter m'/δ where m' is the number of edges in the sparsification.*

Proof. By the way its constructed and Lemma 6.1.1, the oracle returns an admissible dual witness. Since we remove edges with weight less than $\delta W/m'$ and $\beta_p = \alpha = O(W)$, the width parameter is $O(m'/\delta)$. □

Proof of Theorem 10.3.1. From the width parameter, the number of iterations is $\tilde{O}(n/\delta^5)$ and the running time for each iteration is $\tilde{O}(n/\delta^2)$. Therefore, the running

Algorithm 34 Oracle for LP55

Normalize x_{ij} and y_{ij} so that $\sum_{ij} w_{ij}x_{ij} + \sum_{ij} w_{ij}y_{ij} = \alpha$.

Let $l_{ij} = x_{ij}$ for $(i,j) \in E^+$ and $l_{ij} = 1 - y_{ij}$ for $(i,j) \in E^-$.

Round the fractional solution.

if u and v belong to the same cluster for $(u,v) \in E^-$ and $l_{uv} > 2/3$ **then**

  Find the corresponding path p (that violates a constraint).

  Return $\beta_p = \alpha$ and $\beta_{p'} = 0$ for $p' \neq p$.

else

  Return the rounded solution.

end if

time of the algorithm is $\tilde{O}(n^2/\delta^7)$.

The optimal fractional solution for the sparsification is bounded by $OPT + \frac{2\delta W}{\log n}$. The rounded solution has at most $O(\log n)$ times the fractional solution which is at most $O(\log n)OPT + O(\delta)W$. Ignored edges (due to small edge weight) and the sparsification process contributes additional δW and $\frac{2\delta W}{\log n}$ additive error. By summing up all the errors, we obtain the desired result. $\square$

10.4 Correlation Clustering: Maximizing Agreements

In this section, we present a $0.7666(1-\epsilon)$-approximation algorithm for the maximization correlation clustering problem. There are algorithms based on SDP in the RAM model [91, 25]. We apply Swamy's 0.7666-approximation algorithm to the semi-streaming model with the SDP feasibility algorithm by Steurer [90]. The outline of the algorithm is as follows.

1. We sparsify the graph. The sparsification algorithm preserves the optimal value with δW additive error. Since the optimal solution is at least $W/2$, this preserves optimal clustering within $1 \pm \delta$ factor. We then formulate the SDP for the sparsified graph. The following is the SDP formulation.

$$\begin{aligned} \max \quad & \textstyle\sum_{(i,j)\in E^+} w_{ij}(||x_i||^2 + ||x_j||^2 - ||x_i - x_j||^2) \sum_{(i,j)\in E^-} w_{ij}||x_i - x_j||^2 \\ \text{s.t.} \quad & ||x_i||^2 = 1 \quad \text{for all } i \in V \\ & x_i \cdot x_j \geq 0 \quad \text{for all } i, j \in V \end{aligned} \tag{SDP1}$$

2. We solve the SDP with the feasibility algorithm. The feasibility algorithm requires a separation oracle which finds a set of violated constraints. The feasibility algorithm will be explained in Section 2.3.3 and the oracle will be explained in Section 10.4.1.

3. We round the fractional solution with Swamy's algorithm [91] which requires a

proof that the rounding algorithm is robust, i.e., the algorithm still works even if constraint are violated by small amount δ. This is proven in Section 10.4.2.

Summarizing the above, we obtain:

Theorem 10.4.1. *There is an one-pass* $0.7666(1-\epsilon)$*-approximation algorithm for the maximizing agreements problem in the semi-streaming model that runs in* $\tilde{O}(m + \frac{n}{\epsilon^8})$ *time.*

Proof. We need $O(\frac{\log n}{\delta^4})$ iterations since the width parameter $\rho = O(1/\delta)$ (see Lemma 10.4.3) and each iteration of the algorithm takes $\tilde{O}(n/\delta^4)$ (see Lemma 10.4.4). The final rounding algorithm takes $\tilde{O}(m')$ time. Since we have $O(1/\delta)$ guesses of the optimal solution, the total running time is $\tilde{O}(n/\delta^9)$. □

10.4.1 Oracle

We design an oracle for SDP1 in a similar way to the oracles for LP55 and LP52. We find a set of violated constraints. There are entries of $\mathbf{A}$ that correspond to the violated constraints. We assign non-zero values to those entries. Algorithm 35 shows the oracle for SDP1.

The oracle has the following properties which bounds the running time of the algorithm.

Lemma 10.4.2. *Algorithm 35 is* $\delta -$ *separating.*

Algorithm 35 Oracle for SDP1

Let D be the diagonal matrix with with $D_i i = \frac{d_i}{\sum_j d_j} = \frac{d_i}{2W}$. For the separating hyperplane, we only describe non-zero entries.

Let $S_1 = \{i | ||x_i||^2 \geq 1 + \delta\}$ and $\Delta_1 = \sum_{i \in S_1} d_i$.

Let $S_2 = \{i | ||x_i||^2 \leq 1 - \delta\}$ and $\Delta_2 = \sum_{i \in S_2} d_i$.

Let $S_3 = \{(i,j) | x_i \cdot x_j < -\delta\}$ and $\Delta_3 = \sum_{(i,j) \in S_3} w_{ij}$.

if $\Delta_1 \geq \delta\alpha$ **then**

  Let $\mathbf{A}_{ii} = -\frac{d_i}{\Delta_1}$ for $i \in S_1$ and $b = -1$. Return $(\mathbf{A}, b)$.

else if $\Delta_2 \geq \delta\alpha$ **then**

  Let $\mathbf{A}_{ii} = \frac{d_i}{\Delta_2}$ for $i \in S_2$ and $b = 1$. Return $(\mathbf{A}, b)$.

else if $\Delta_3 \geq \delta\alpha$ **then**

  Let $\mathbf{A}_{ij} = \frac{w_{ij}}{\Delta_3}$ for $(i,j) \in S_3$ and $b = 0$. Return $(\mathbf{A}, b)$.

end if

Ignore all nodes in S_1 and S_2 and all edges in S_3. Let $\mathbf{C}'$ be the matrix that corresponds to the objective function of the modified graph.

if $\mathbf{C}' \circ \mathbf{X} < (1 - 4\delta)\alpha$ **then**

  Let $\mathbf{A} = \frac{1}{\alpha}\mathbf{C}'$ and $b = 1 - 3\delta$. Return $(\mathbf{A}, b)$.

else

  Round $\mathbf{X}$ and return the rounded solution.

end if

Proof. For line 6, $\mathbf{A} \circ \mathbf{X} \leq \sum_{i \in S_1} -\frac{d_i(1+\delta)}{\Delta_1} = -1 - \delta$ since $||x_i||^2 \geq 1 + \delta$ for all $i \in S_1$. On the other hand, for any feasible $\mathbf{X}'$, $||x_i||^2 = 1$ for all i. Hence $\mathbf{A} \circ \mathbf{X}' = \sum_{i \in S_1} \frac{-d_i}{\Delta_1} = -1$. At line 8 and line 10, we have almost identical proofs.

For line 14, we do not use the violated constraints. Instead we use $\mathbf{C}'$ to construct $\mathbf{A}$. We show that $\mathbf{C}' \circ \mathbf{X}' \geq (1 - 3\delta)\alpha$. We start from the fact that $\mathbf{C} \circ \mathbf{X}' \geq \alpha$. By removing all nodes in S_1, we remove all edges adjacent to the removed nodes. The total weight of removed edges are bounded by $\Delta_1 \leq \delta\alpha$. Similarly, we lose at most $\delta\alpha$ for S_2 and S_3. Hence, the difference between $\mathbf{C}' \circ \mathbf{X}'$ and $\mathbf{C} \circ \mathbf{X}'$ is bounded by $3\delta\alpha$. Therefore, $\mathbf{A} \circ \mathbf{X} \leq 1 - 4\delta$ while $\mathbf{A} \circ \mathbf{X}' \geq 1 - 3\delta$ for $\mathbf{X}' \in \mathcal{X}$. □

Lemma 10.4.3. *Algorithm 35 is $O(1/\delta)$-bounded.*

Proof. We prove that for any $\mathbf{X}$, $|\mathbf{A} \circ \mathbf{X}| \leq O(\frac{1}{\delta})\mathbf{D} \circ \mathbf{X}$. For line 6 (and line 8), the proof is straight forward. $\mathbf{A}$ is also a diagonal matrix where $|\mathbf{A}_{ii}| = \frac{d_i}{\Delta_1} \leq O(\frac{1}{\delta})\frac{d_i}{\alpha}$. On the other hand, $\mathbf{D}_{ii} = \frac{d_i}{2W}$. From the fact that $\alpha \geq W/4$, we have $|\mathbf{A}_{ii}| = O(\frac{1}{\delta})\mathbf{D}_{ii}$ which proves that $|\mathbf{A} \circ \mathbf{X}| \leq O(\frac{1}{\delta})\mathbf{D} \circ \mathbf{X}$. The proof is identical for line 8.

For line 10 and line 14, we use the fact that $x_i \cdot x_j \leq ||x_i||^2 + ||x_j||^2$. By replace each $\mathbf{A}_{ij} x_i \cdot x_j$ by $|\mathbf{A}_{ij}|(||x_i||^2 + ||x_j||^2)$ we obtain a diagonal matrix $\mathbf{A}'$. We now compare $\mathbf{A}'_{ii}$ with $\mathbf{D}_{ii}$. For line 10, $\mathbf{A}'_{ii} = \frac{1}{\Delta_3} \sum_{(i,j) \in S_3} 2w_{ij} = O(\frac{\delta d_i}{\alpha})$. By the same argument as line 6, we have $|\mathbf{A} \circ \mathbf{X}| \leq O(\frac{1}{\delta})\mathbf{D} \circ \mathbf{X}$. For line 14, let d'_i be the degree in the modified graph. Then, $\mathbf{A}'_{ii} = \frac{O(1)d'_i}{\alpha}$. Therefore, $|\mathbf{A} \circ \mathbf{X}| \leq O(1)\mathbf{D} \circ \mathbf{X}$.

Summarizing, Algorithm 35 is $O(1/\delta)$-bounded. □

Lemma 10.4.4. *Approximating* $\mathbf{X}$ *and the execution of Algorithm 35 can be done in* $\tilde{O}(n/\delta^4)$ *time.*

Proof. In order to execute Algorithm 35 correctly, we have to approximate diagonal entries of $\mathbf{X}$ with $1 \pm O(\delta)$ additive error which can be done with $d = O(\frac{\log n}{\delta^2})$. In addition, for edges (i, j) with $1-\delta \leq ||x_i||^2, ||x_j||^2 \leq 1+\delta$, $x_i \cdot x_j$ must be approximated with $O(\delta)$ additive error which can be done with $d = O(\frac{\log n}{\delta^2})$. So $d = O(\frac{\log n}{\delta^2})$ is enough. Also, $r = O(\log \frac{1}{\delta})$ is enough from the same error bounds.

There are at most $O(m' + n)$ non-zero entries on $\mathbf{A}$ since $\mathbf{A}_{ij} \neq 0$ only if (i, j) is in the sparsified graph. Therefore, $T_M = O(m' + n) = \tilde{O}(n/\delta^2)$ and it takes $\tilde{O}(n/\delta^4)$ time to approximate $\mathbf{X}$. Once $\mathbf{X}$ is approximated, the execution of Algorithm 35 can be also done in $O(m')$ time which is dominated by $\tilde{O}(n/\delta^4)$. □

10.4.2 Rounding the Fractional Solution

The proof of the next lemma is routine for SDP based approximation — it is a simple robustness property. However this lemma has to be shown for every oracle for the overall framework to work.

Lemma 10.4.5. *If Algorithm 35 returns a rounded solution, it returns a clustering with agreements at least* $0.7666(1 - O(\delta))\alpha$.

Proof. We show that the rounding algorithm returns a clustering with agreements at least $0.7666(1 - O(\delta))\mathbf{C}' \circ \mathbf{X}$. Combined with the fact that $\mathbf{C}' \circ \mathbf{X} > (1 - \delta)\alpha$, we obtain the desired result.

Since we deal with $\mathbf{C}'$ instead of $\mathbf{C}$, we can ignore all nodes and edges in S_1, S_2, and S_3. We first change the vectors in $\mathbf{X}$ to be unit vectors. Since all remaining vectors has length between $1-\delta$ and $1+\delta$, this only changes the objective value by $O(\delta w_{ij})$ for each edge and hence, the total decrease is bounded by $O(\delta W) = O(\delta\alpha)$.

We then change the objective value of edges (i,j) with $-\delta < x_i \cdot x_j < 0$. This decreases the objective value by at most δw_{ij} for each negative edge. Again, the objective value decreases by at most $O(\delta\alpha)$.

We then use the Swamy's rounding algorithm [91] which obtains 0.7666 approximation factor. One problem is that the constraint $x_i \cdot x_j \geq 0$ is not satisfied for some edges. However, the rounding algorithm is based on random hyperplanes and the probability that x_i and $x_j j$ are split by a hyperplane only increases as $x_i \cdot x_j$ decreases. For positive edges, we already account its objective value as 0 (always disagrees). For the negative edges, the probability we put i and j in different clusters only increases by having negative $x_i \cdot x_j$. Note that we already changed the objective value so that $x_i \cdot x_j = 0$. Therefore, we obtain a clustering that agrees at least $0.7666(1-O(\delta))$ fraction of $\mathbf{C}' \circ \mathbf{X}$. $\square$

Bibliography

[1] R. Adler and Y. Azar. Beating the logarithmic lower bound: Randomized preemptive disjoint paths and call control algorithms. *SODA*, pages 1–10, 1999.

[2] K. Ahn and S. Guha. A fully-polynomial (near) linear time approximation schemes for b– matching problems in general graphs. *Manuscript*, 2013.

[3] K. Ahn and S. Guha. Primal dual or dual primal? iterative algorithms for linear programming. *Manuscript*, 2013.

[4] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *ICALP (2)*, pages 328–338, 2009.

[5] K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *ICALP (2)*, pages 526–538, 2011.

[6] K. J. Ahn and S. Guha. Multicut and correlation clustering in the semi-streaming model. *Manuscript*, 2011.

[7] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *SODA*, 2012.

[8] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: Sparsification, spanners and subgraphs. *Proc. of PODS*, 2012.

[9] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), 2008.

[10] R. P. Anstee. A polynomial algorithm for b-matchings: An alternative approach. *Information Processing Letters*, 24(3):153 – 157, 1987.

[11] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[12] S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007.

[13] B. Bahmani, A. Goel, and K. Munagala. Efficient primal dual algorithms for mapreduce. *Manuscript*, 2012.

[14] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[15] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[16] S. Baswana. Faster streaming algorithms for graph spanners. *Manuscript, available at* `http://arxiv.org/abs/cs/0611023`, 2006.

[17] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.

[18] J. D. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. In *STOC*, pages 255–262, 2009.

[19] A. A. Benczúr and D. R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[20] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. *Proc. STOC*, 2007.

[21] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. P. Woodruff. Transitive-closure spanners. *SIAM J. Comput.*, 41(6):1380–1425, 2012.

[22] A. L. Buchsbaum, R. Giancarlo, and J. Westbrook. On finding common neighborhoods in massive graphs. *Theor. Comput. Sci.*, 1-3(299):707–718, 2003.

[23] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS*, pages 253–262, 2006.

[24] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, pages 328–335, 2007.

[25] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

[26] M. Charikar, T. Leighton, S. Li, and A. Moitra. Vertex sparsifiers and abstract rounding algorithms. In *FOCS*, pages 265–274, 2010.

[27] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[28] W. Cook. On box totally dual integral polyhedra. In *Mathematical Programming*, pages 48–61, 1986.

[29] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.

[30] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

[31] W. H. Cunningham and A. B. Marsh. A primal algorithm for optimum matching. *Polyhedral Combinatorics, Mathematical Programming Studies*, 8:50–72, 1978.

[32] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[33] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.

[34] R. Duan and S. Pettie. Approximating maximum weight matching in near-linear time. In *FOCS*, pages 673–682, 2010.

[35] R. Duan, S. Pettie, and H.-H. Su. Scaling algorithms for approximate and exact maximum weight matching. In *Arxiv http://arxiv.org/abs/1112.0790*, 2011.

[36] R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *SODA*, pages 1413–1424, 2012.

[37] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.

[38] S. Eggert, L. Kliemann, P. Munstermann, and A. Srivastav. Bipartite graph matchings in the semi-streaming model. *Technical Report, Institut für Informatik, available at http://arxiv.org/abs/1104.4058*, 2011.

[39] S. Eggert, L. Kliemann, and A. Srivastav. Bipartite graph matchings in the semi-streaming model. In *ESA*, pages 492–503, 2009.

[40] M. Elkin and J. Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[41] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[42] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *Proc. of STACS*, pages 347–358, 2010.

[43] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

[44] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.

[45] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geometry Appl.*, 18(1/2):3–28, 2008.

[46] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80, 2011.

[47] Z. Füredi. Maximum degree and fractional matchings in uniform hypergraphs. *Combinatorica*, 1(2):155–162, 1981.

[48] Z. Füredi, J. Kahn, and P. D. Seymour. On the fractional matching polytope of a hypergraph. *Combinatorica*, 13(2):167–180, 1993.

[49] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. *Proc. STOC*, pages 448–456, 1983.

[50] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *SODA*, pages 434–443, 1990.

[51] S. Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.

[52] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996.

[53] F. R. Giles and W. R. Pulleyblank. Total dual integrality and integer polyhedra. *Linear Algebra and Applications*, 25:191–196, 1979.

[54] A. Goel, M. Kapralov, and I. Post. Single pass sparsification in the streaming model with edge deletions. *Manuscript, available at* `http://arxiv.org/abs/1203.4900`, 2012.

[55] R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[56] M. Grotschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.

[57] R. Hariharan, T. Kavitha, and D. Panigrahi. Efficient algorithms for computing all low s-t edge connectivities and related problems. *Proc. SODA*, 2007.

[58] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams, 1998.

[59] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[60] S. Hougardy. Linear time approximation algorithms for degree constrained subgraph problems. *Research Trends in Comb. Opt., Springer*, pages 185–200, 2008.

[61] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[62] P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM New York, NY, USA, 2005.

[63] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *COCOON*, pages 710–716, 2005.

[64] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58, 2011.

[65] B. Kalantari and A. Shokoufandeh. Approximation schemes for maximum cardinality matching. Technical Report LCSR-TR-248, Laboratory for Computer Science Research, Department of Computer Science. Rutgers University, 1995.

[66] M. Kapralov. Better bounds for matchings in the streaming model. *SODA*, 2013.

[67] D. R. Karger. Random sampling in cut, flow, and network design problems. In *STOC*, pages 648–657, 1994.

[68] J. A. Kelner, G. L. Miller, and R. Peng. Faster approximate multicommodity flow using quadratically coupled flows. *STOC*, pages 1–18, 2012.

[69] C. Koufogiannakis and N. E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. *DISC*, pages 221–238, 2009.

[70] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, pages 85–94, 2011.

[71] F. T. Leighton and A. Moitra. Extensions and limits to vertex sparsification. In *STOC*, pages 47–56, 2010.

[72] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[73] A. N. Letchford, G. Reinelt, and D. O. Theis. A faster exact separation algorithm for blossom inequalities. *Proceedings of IPCO, LNCS 3064*, pages 196–205, 2004.

[74] A. McGregor. Finding graph matchings in data streams. In *APPROX-RANDOM*, pages 170–181, 2005.

[75] J. Mestre. Greedy in approximation algorithms. *ESA*, pages 528–539, 2006.

[76] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.

[77] A. Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *FOCS*, pages 3–12, 2009.

[78] M. Monemizadeh and D. P. Woodruff. 1-pass relative-error $\mathrm{l_p}$-sampling with applications. In *SODA*, pages 1143–1160, 2010.

[79] M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: towards a flexible software design. *J. Exp. Algorithmics*, 4, 1999.

[80] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[81] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[82] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[83] S. Pettie and P. Sanders. A simpler linear time 2/3-epsilon approximation for maximum weight matching. *Inf. Process. Lett.*, 91(6):271–276, 2004.

[84] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[85] R. Preis. Linear time 1/2 -approximation algorithm for maximum weighted matching in general graphs. In *Proceedings of the 16th annual conference on Theoretical aspects of computer science*, STACS'99, pages 259–269, Berlin, Heidelberg, 1999. Springer-Verlag.

[86] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[87] D. Shmoys and D. Williamson. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[88] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *STOC*, pages 563–568, 2008.

[89] D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

[90] D. Steurer. Fast SDP algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010.

[91] C. Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *SODA*, pages 526–527, 2004.

[92] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[93] D. E. D. Vinkemeier and S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms*, 1(1):107–122, 2005.

[94] D. P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *FOCS*, pages 389–398, 2006.

[95] M. Zelke. Weighted matching in the semi-streaming model. *Proc. of STACS*, pages 669–680, 2008.